우리식조작체계문고

JAVA 자료기지 프로그람작성법

교육성교육정보쎈터

주체97(2008)

차 례

머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《프로그람을 개발하는데서 기본은 우리 식의 프로그람을 개발하는것입니다.》

(《**김정일**선집》 제15권, 196페지)

위대한 령도자 **김정일**동지의 현명한 령도에 의하여 최근년간 우리 나라에서는 콤퓨터기술이 급속히 발전하고 인민경제 여러 부문에 광범히 도입되여 그 경제적효과성이 높아지고있다.

전국적범위에서 콤퓨터망이 형성되여 인민경제의 정보화가 적극 추진되고있는 현실적요구에 맞게 능률적인 프로그람들을 개발하자면 망기반의 프로그람작성언어인 Java를 리용하여 자료기지를 설계구축하고 합리적으로 리용하는것이 매우 중요한 문제로 나선다.

더우기 우리 식 조작체계 《붉은별》이 개발된 조건에서 과학연구 및 경영활동에 필요한 모든 프로그람작성을 능란하게 하려면 Java를 리용하여 방대한 자료를 분류하고 해석하며 요구에 따라 제때에 추출하는 능력을 가져야 한다.

이 책에서는 생산과 경영활동에서 가장 많이 리용되고있는 관계형자료기지관리체계의 개념으로부터 자료기지설계방법과 그 응용, SQL문의 작성 그리고 JDBC에 기초한 2층구성방식과 Apache/Tomcat기초의 웨브봉사기를 중심으로 구축한 3층구성방식, 자료기지와 XML을 함께 리용하는 방법 등 Java자료기지프로그람을 작성하는데서 기본적인 문제들을 서술하였다.

또한 하나의 완전한 업무체계를 구축하는 과정을 통하여 Java와 자료기지를 배우는데 도움이 되도록 여러가지 실례들을 주었다.

우리 식 조작체계에 애착을 가지고 열심히 배워나가는 독자들에게 조금이나마 도움이 되기를 바라마지 않는다.

제 1 장. 관계형자료기지

이 장의 목적은 관계형자료기지관리체계(Relational Database Management System: RDBMS)의 원리적인 개념들을 설명함으로써 이 책 전반을 리해하기 위한 기본토대를 구축하는데 있다.

제1절. 관계형자료기지관리체계

자료기지(database)란 후에 찾아볼수 있도록 기계가 읽을수 있는 형식으로 저장된 의미있는 자료의 모임이다. 이 정의는 상당히 통속적인것으로서 자료기지의 구조나 방법에 대해서는 말하지 않고있다. 이 정의에 따르면 파일이나 파일들의 모임도 자료기지로 간주될수 있다. 보다 엄밀히 말하면 자료기지는 자기의 자료를 관리하는 체계부분을 포함하여야 한다. 이렇게 놓고 보면 자료기지는 단순한 파일들의 모임이 아니라 하나의 완전한 체계여야 한다.

자료기지에서 자료의 관리를 제공하는 체계부분을 **자료기지관리체계**(Database Management System)라고 한다. 실제적인 자료기지관리체계는 자료를 관리하고 주고받을수 있는 능력과 자료의 물리적인 저장장치와의 결합이다. 이러한 체계는 다음의 과제들을 지원하여야 한다.

- 론리적인 자료구조의 작성과 관리
- 자료의 기입과 검색
- 론리적이며 일관한 수법에 의한 자료조작
- 믿음성있는 자료저장

현대적인 관계형자료기지가 개발되기전에 여러가지 방법들이 수많이 시도되였지만 그것들은 대체로 단순하고 특수한 목적을 위하여 설계된 전용의 자료저장체계였다.

오늘날의 자료기지는 일반적으로 관계형자료기지이다. 그러므로 우리는 여기서 관계형자료기지관리체계에 대하여 보기로 한다.

1.1.1. 관계모형

자료기지기술을 크게 전진시킨것은 **관계형자료기지모형**(relational database model)의 개발이였다. 관계형자료기지는 1960년대말 수학자 코드(E.F. Codd)의 연구활동에서부터 유래되였다. 그의 모형은 수학에서의 모임론(set theory)과 술어론리(predicate logic)에 기초하고있다. 그는 1970년 6월에 발표한 론문에서 relation, attribute, tuple이라는 용어를 사용하였는데 그것들은 오늘날 표(table), 렬(column), 행(row)과 같은 보다 일반적인 용어로 불리우고있다. 자료기지전문가들은 여전히 relation,

attribute, tuple이라는 용어를 사용하지만 일반사용자들은 그 보다도 표, 렬, 행이라는 용어를 더 잘 사용한다.

코드의 관계모형은 관계형자료기지의 3가지 기본요구 즉 구조(structure), 완정성(integrity), 자료조작(data manipulation)을 다 가지고있다. 관계모형의 기본원리는 다음과 같다.

- 관계형자료기지는 순서화되지 않은 수많은 표들로 이루어진다.
- 이 표들의 구조는 자료를 저장하는 물리적인 자료저장매체와 독립이다.
- 표의 내용은 비수속적인 연산들로 조종할수 있는데 결과는 표로 돌려진다.

이 관계모형에 따르면 사용자는 자료의 물리적구조를 몰라도 자료기지의 자료에 접근하고 조작할수 있다. 즉 사용자는 파일참조나 지적자를 리용하여 자료에 접근하는것이 아니라 일반적인 표형식의 구조를 통하여 자료를 다룬다. 관계모형에서는 사용자에게 현시되는 자료의 론리적인 뷰와 체계에 저장된 자료의 물리적구조가 명백히 구별된다.

코드는 표대신에 관계라는 말을 사용하였지만 사실상 그의 모형은 간단한 표구조에 기초하고 있다. 매개 표는 하나이상의 행들로 이루어지며 매개 행은 표의 렬에 대응하는 많은 마당(field)들을 포함하고있다.

코드가 정의한 표형식의 구조는 단순하고 리해하기 쉽다. 또한 대부분의 자료형들을 임의의 구조로 충분히 표현할수 있다. 표형식의 다른 하나의 우점은 결과를 표형식으로 표현하는 잘 정의된 수학적인 연산들로 자료를 처리할수 있다는것이다. 이 수학적인 연산들은 고수준언어로 쉽게 실현할수 있다. 실제상 코드의 규칙도 이러한 목적으로부터 고수준언어가 관계형자료기지관리체계에 병합되여야 한다는것을 요구하고있다. 그 언어는 뒤에서 배우게 될 구조화질문언어(Structured Query Language: SQL)로 발전하였다.

고수준언어를 리용하여 론리적준위에서 자료를 조작할수있게 한것은 관계형자료기지의 중요한 특징이다. 고수준언어를 리용하면 사용자가 자료의 물리적구조대신 그의 속성에 기초하고있는 표를 통하여 자료를 삽입하거나 검색하게됨으로 자료조작의 추상화가 충분히 보장되게 된다. 실례로 사용자가 어떤 사람에 대한 개인정보를 찾으려고 할 때 디스크의 어느 위치에 저장된 자료를 검색하라고 요구하는것이 아니라 자료의 속성 즉 사람의 이름이나 공민증번호에 의해 그 사람에 대한 자료를 검색할수 있게 한다.

이 방법의 보다 큰 우점은 사용자가 론리적방법으로 자료요청을 할 때 자료관리체계는 자료저장체계의 물리적처리에 비하여 고도로 최적화된 자료요청결과를 제공해준다는것이다. 다시말하여 자료관리체계는 물리적인 조작에서 론리적인 조작을 분리시킴으로써 사용자에게 보다 친근하면서도 높은 효률성을 제공해준다.

1.1.2. 표

관계형자료기지에서 모든 자료는 반드시 론리적준위에서 표의 값으로 명백하게 표현되여야 한다. 다시말하여 표는 관계형자료기지관리체계의 기초이다. 관계모형에서 표는 현실세계의 객체 혹은 사건들의 모임을 나타낸다. 하나의 표는 학생 혹은 주문자와 같은 단일한 류형의 객체들의 모임이다.

모든 관계형자료기지는 다음의 설계개념에 의거하고있다.

- 관계형자료기지에서 모든 자료는 론리적준위에서 표의 값으로 명백하게 표현된다.
- 표의 매 세포는 단일한 자료항목에 대한 값을 가진다.
- 같은 렬에 놓이는 세포들은 류사한 항목모임의 요소이다.
- 같은 행에 놓이는 세포들은 련관된 항목집단의 요소이다.
- 모든 표는 매개 행을 유일하게 식별하는 하나 혹은 여러개의 렬로 이루어진 열쇠를 정의한다.

표 1-1은 관계형자료기지에서 이름과 주소로 이루어진 전형적인 표이다.표의 매 행에는 특정한 주문자에 대한 련관된 자료들이 포함되게 되며 매 렬에는 이름이나 거주지와 같이 동일한 종류의 자료들이 포함되며 매세포에는 특정한 주문자에 대한 한가지 종류의 자료 하나만이 포함되게 된다.

ID렬은 다른 렬들과 약간 다르다. ID렬은 주어진 대상에 대한 정보가 아니라 그 대상에 대하여 체계가 할당한 유일한 식별자이다. 이 식별자를 **기본열쇠**(primary key)라고 부른다.

표 1-1. 주문자표(Customers)

Customer_ ID(주문자ID)	Family_ Name(성)	Personal_ Name(이름)	Sex (성별)	State (도)	City (시)	District (구역/군)	Dong (동/리)	Neighbor (인민반)
100	김	성철	남	평양시	평양	중구역	교구동	32
101	김	은희	녀	평양시	평양	보통강구역	대보동	17
102	김	철	남	함경남도	함흥		사포동	23
103	박	성호	남	평안남도	남포	와우도구역	천리마동	65
104	강	권혁	남	평양시	평양	중구역	류성동	23
105	하	성진	남	황해북도	사리원		경암동	28
106	최	영실	녀	강원도	원산		석현동	26
107	리	성민	남	평안북도	신의주		역전동	33
108	오	성철	남	평양시	평양	평천구역	안산2동	56

이 표는 관계형자료기지에서 가장 중요한 두가지 요구를 보여주고있다.

- 관계형자료기지에서 모든 자료는 론리적준위에서 표안의 값으로 표현된다.
- 모든 자료요소에는 표이름, 기본열쇠이름, 기본열쇠의 값, 렬이름의 조합을 리용하여 접근할수 있다.

행들의 순서가 의미를 가지지 않는다는것은 실례로부터 명백하다. 실레에서 보는바와 같이 표는 매 행이 ID에 따라 순서화되였든 도별자모순에 따라 정렬되였든 상관없이 같은 정보를 포함한다.

자료기지관리체계는 자료기지자체에 대한 설명이나 목록도 론리적준위에서 표의 값으로 저장하며 관계형언어는 표구조로 저장된 자료에 작용하는것과 꼭 같은 방법으로 자료기지설계에 작용할수 있다. 관계형자료기지관리체계들은 체계표들을 통하여 이것을 실현한다. 이 표들에는 자료기지접근에 사용하는것과 꼭 같은 자료기지도구들을 리용하여 접근할수 있다.

그림 1-1은 이 책에서 고찰하는 《주문체계》자료기지의 표들을 현시하는 SQL Sever를 보여주고있다. 체계표들은 보통 SQL Sever에서 소문자로 현시되므로 사용자용의 표이름들은 대문자를 사용하는것이 좋다. 실례로 체계표 syscolumns는 이 자료기지의 표들에 들어있는 모든 렬들에 대한 정보를 보관하고있는 표이다.

Name	Owner	Type	Create Date
CONTACT	dbo	User	2005/8/2 3:11:34 오후
CONTACT_INFO	dbo	User	2005/7/29 3:15:01 오후
CUSTOMERS	dbo	User	2005/8/1 2:41:33 오후
dtproperties	dbo	System	2005/7/22 7:19:57 오후
EMPLOYEES	dbo	User	2005/8/1 9:37:33 오전
INVENTORY	dbo	User	2005/7/29 7:07:08 오후
ITEMS	dbo	User	2005/7/29 3:14:09 오후
ORDERED_ITEMS	dbo	User	2005/7/29 3:13:23 오후
ORDERS	dbo	User	2005/7/29 7:22:04 오후
PWCHECK	dbo	User	2005/8/1 2:53:15 오후
STOCK	dbo	User	2005/8/1 6:22:01 오후
syscolumns	dbo	System	2000/8/6 1:29:12 오전
syscomments	dbo	System	2000/8/6 1:29:12 오전
sysdepends	dbo	System	2000/8/6 1:29:12 오전
sysfilegroups	dbo	System	2000/8/6 1:29:12 오전
sysfiles	dbo	System	2000/8/6 1:29:12 오전
sysfiles1	dbo	System	2000/8/6 1:29:12 오전
sysforeignkeys	dbo	System	2000/8/6 1:29:12 오전
sysfulltextcatalogs	dbo	System	2000/8/6 1:29:12 오전
sysfulltextnotify	dbo	System	2000/8/6 1:29:12 오전
sysindexes	dbo	System	2000/8/6 1:29:12 오전
sysindexkeys	dbo	System	2000/8/6 1:29:12 오전
sysmembers	dbo	System	2000/8/6 1:29:12 오전
sysobjects	dbo	System	2000/8/6 1:29:12 오전
syspermissions	dbo	System	2000/8/6 1:29:12 오전
sysproperties	dbo	System	2000/8/6 1:29:12 오전
sysprotects	dbo	System	2000/8/6 1:29:12 오전
sysreferences	dbo	System	2000/8/6 1:29:12 오전
systypes	dbo	System	2000/8/6 1:29:12 오전
sysusers	dbo	System	2000/8/6 1:29:12 오전

그림 1-1. SQL Sever가 생성한 사용자표들과 체계표들

Null값

자료기지를 만들 때 실천적으로 사용자가 자료요소의 값을 알수 없다든가 혹은 적용할수 있는 값을 가지지 않는것과 같은 경우들에 부닥치게 된다.

우선 자료의 값을 알수 없는 경우는 두가지로 나누어 볼수 있는데 그 값이 존재하지만 현재 없는 경우와 그 값이 존재하는지조차 알수 없는 경우이다. 례를 들어 종업원의 이름값이 Null이라면 값이 빠진것이고 그의 취미를 가리키는 렬값이 Null이라면 실제로 어떤 취미가 있는지 알지 못할수 있다.

다음으로 적용할 값을 가지지 않는 두번째 경우의 실례를 들어보자. 종업원명단에 자식렬이 있는 경우 미혼자에게는 이 자식관계가 해당되지 않기때문에 마땅히 Null값을

가지게 된다.

관계형자료기지관리체계는 현재 없거나 적용할수 없는 정보에 대한 표현방법을 지원하여야 한다. 그것은 의도적인것으로서 보통의 다른 값들과는 구별되며 자료형과 독립이다. 다시말하여 관계형자료기지는 사용자가 어떤 마당에 대한 값을 알수 없거나 적용할수 없는 경우 그것을 의미하는 NULL값을 삽입할수 있어야 한다.(표 1-2)

표 1-2. 표에 NULL값의 삽입

Customer_ID	Family_Name	Personal_Name	Sex	State	City	District	Dong	Neighbor
100	김	성철	남	평양시	평양	중구역	교구동	32
101	김	은희	녀	평양시	평양	보통강구역	대보동	17
102	김	철	남	함경남도	함흥	<NULL>	사포동	23
103	박	성호	남	평안남도	남포	와우도구역	천리마동	65
104	강	권혁	남	평양시	평양	중구역	류성동	23
105	하	성진	남	황해북도	사리원	<NULL>	경암동	28

Null값에 대한 지원은 관계형자료기지관리체계가 표준적인 연산과정에 계획된 방법으로 그것들을 처리한다는것을 의미한다.

1.1.3. 열쇠

기본열쇠

모든 표는 매개 행에 대하여 유일한 값을 가지는 임의의 렬 혹은 렬들의 조합으로 이루어지는 오직 하나의 기본열쇠를 가질수 있다.

대부분의 관계형자료기지들에서는 기본열쇠가 없어도 표를 작성할수 있지만 사용자가 기본열쇠를 할당하지 않는 경우 그 표를 사용하기 곤난하다. 그 리유는 관계형자료기지의 장점의 하나가 표들을 서로 련결하는 능력인데 표들사이의 련결에 바로 기본열쇠가 리용되기때문이다.

기본열쇠는 한개의 렬로 된 단일한것도 있고 둘이상의 렬로 이루어진 복합적인것도 있다. 기본열쇠를 위한 렬 혹은 렬들의 조합을 어떻게 선택하는가 하는것은 다음의 인자들을 고려하여 결정한다.

- 열쇠접근의 효률을 높이기 위하여 가능한 제일 적은 수의 렬을 리용하여야 한다.
- 기본열쇠를 변경하면 표들사이의 련결이 파괴되기때문에 변하지 않는 렬이나 렬들의 조합을 리용하여야 한다.
- 단순하고 리해하기 쉬운 렬이나 렬들의 조합을 리용하여야 한다.

실천에서 가장 일반적으로 리용되는 열쇠의 류형은 기본열쇠로 리용할 목적으로 특별히 작성한 유일한 옹근수들의 렬이다. 유일한 옹근수들은 표에서 행식별자 혹은 ID의 역할을 한다.

외부열쇠

외부열쇠(Foreign key)는 표에서 다른 표의 기본열쇠를 참조하는데 리용되는 렬이다. 만일 자료기지가 오직 하나의 표나 련관되지 않은 여러개의 표들로 이루어졌다면 기본열쇠의 리용이 많지 않을것이다. 기본열쇠는 련관된 여러개의 표들과 함께 작업할 때 중요하게 리용된다. 례를 들어 백화점들에는 주문자표와 함께 상품목록표, 주문항목표, 주문표 등이 있을수 있다. 표 1-3에서는 상품목록표를 보여주었다.

표 1-3. 상품목록표(Inventory)

Item_Number (품목번호)	Name (상품명)	Description (종류)	Qty (수량)	Cost (원가)
1001	고려인삼술	주류	178	1.95
1002	평양술	주류	97	1.87
1003	백두산들쭉술	주류	103	2.05
1004	강계인풍술	주류	15	0.98
1005	대동강맥주	청량음료	217	0.26
1006	평양맥주	청량음료	162	0.17
1007	룡천사이다	청량음료	276	0.13
1008	모란과자	당과류	144	0.31
1009	살구씨향과자	당과류	96	0.47
1010	박하사탕	당과류	84	0.25

상품목록표에서는 품목번호(Item_Number)렬이 기본열쇠이다.

주문자가 상품을 주문할 때에는 두개의 추가적인 표들을 리용하게 된다. 첫번째 표는 매 주문마다에 주문된 상품항목과 량들을 기록하는 주문항목표이다.(표 1-4)

표 1-4. 주문항목표(Ordered_Items)

ID	Order_Number (주문번호)	Item_Number (품목번호)	Qty (수량)
5000	2	1001	2
5001	2	1004	1
5002	2	1005	4
5003	2	1010	6
5004	3	1006	4
5005	3	1009	2
5006	4	1002	5
5007	4	1003	2
5008	5	1006	3
5009	5	1007	1
5010	5	1008	2

주문항목표는 기본열쇠외에 두개의 외부열쇠를 가지고있다. 그것은 상품목록표의 기본열쇠인 품목번호와 주문표의 기본열쇠인 주문번호이다. 표 1-5에는 주문표를 보여주었다.

표 1-5. 주문표(Orders)

Order_Number (주문번호)	Customer_ID (주문자ID)	Order_Date (주문날자)	Ship_Date (수송날자)
2	101	2007-04-05	2007-04-07
3	102	2007-04-09	2007-04-11
4	104	2007-04-09	2007-04-11
5	100	2007-04-10	2007-04-12
6	107	2007-04-12	2007-04-14
7	106	2007-04-14	2007-04-16

주문표는 주문자의 주문을 정의한 모든 정보들을 포함하고있다. 기본열쇠는 주문번호(Order_Number)렬이고 외부열쇠는 주문자를 식별하기 위한 주문자표의 주문자ID렬이다.

표를 설계할 때 될수록 자료의 중복을 피해야 한다. 다시말하여 어떤 자료항목이 여러곳에 보관되지 말아야 하며 매 자료토막은 해당한 표에 단일한 행으로 보관되여야 한다.

열쇠들이 어떻게 리용되는가를 보기로 하자. 례를 들어 주문번호 3에 대한 주문자정보를 알려면 주문자표에서 기본열쇠(Customer_ID)가 102인 행을 찾아보면 된다. 류사하게 주문항목표에서는 주문번호 3에 주문된 상품들의 품목번호가 1006, 1009이고 수

량은 각각 4, 2라는것을 알수 있다. 또한 상품목록표에서 그 품목번호들을 조사하면 어떤 상품들을 주문했는가를 알수 있다.

이 표들에 있는 정보를 결합하여 우리는 주문번호 3은 주문자ID가 102이고《김철》이라는 이름을 가진 주문자가 2007년 4월 9일에 주문한것이라는것, 그가 주문한 상품들은 품목번호가 1006인 평양맥주 4상자와 품목번호가 1009인 살구씨향과자 2지함이라는것, 주문된 상품들은 2007년 4월 11일에 실어보낸다는것 등을 알수 있다. 이 정보는 다음과 같은 SQL지령을 리용하여 다양한 열쇠들을 일치시켜 얻을수도 있다.

```
SELECT c.Family_Name, c.Personal_Name, i.Name, oi.Qty
FROM Customers c, Orders o, Ordered_Items oi, Inventory i
WHERE o.Order_Number = 3 AND
   c.Customer_ID = o.Customer_ID AND
   i.Item_Number = oi.Item_Number AND
   o.Order_Number = oi.Order_Number;
```

여기서 보여준 SELECT지령과 같은 SQL지령들은 다음 장에서 학습하게 된다.

1.1.4. 관계

우에서 론의된것처럼 열쇠들은 자료기지에서 각이한 표들사이의 관계를 모형화하기 위하여 정의된다. 표들이 관계되는 방식은 3가지이다.

- 1:1
- 1:n
- m:n

1:1관계

첫 표의 모든 행들이 두번째 표에서 오직 하나의 행과 대응하는것을 1:1관계라고 한다. 이러한 관계는 보통 보안을 목적으로 각이한 류형의 자료들을 분리하기 위하여 작성된다. 실례로 사람들은 신용카드자료와 같은 기밀정보들을 보다 제한된 정보로 분리하여 보관하려고 한다.

1:1관계를 가진 표를 작성하는 다른 리유는 자료기지의 실현을 간단히 하기 위해서이다. 례를 들어 여러개의 양식들을 포함하는 웨브응용프로그람을 만들 때 매 양식마다 표를 따로 리용할수도 있다.

표를 1:1관계를 가진 보다 작은 부분들로 분해하는 또 다른 리유는 성능개선 즉 자료기지체계가 지원하는 최대렬수와 같은 자료기지체계의 고유한 제한성들을 극복하기 위해서이다.

1:1관계로 련관된 표들은 항상 꼭 같은 기본열쇠를 가진다. 이것은 련관된 표들이 함께 질문될 때 결합을 실시하는데 리용된다.

1:n관계

첫번째 표의 모든 행들이 두번째 표에서 령, 하나 혹은 많은 대응하는 행들을 가질수 있는 관계를 1:n관계라고 한다. 그러나 두번째 표의 모든 행들은 첫번째 표에서 꼭 하나의 대응하는 행을 가진다. 앞에서 본 주문표와 주문항목표사이의 관계가 바로 1:n관계의 한가지 실례이다. 여기서 하나의 주문은 여러개의 주문상품들과 대응된다.

m:n관계

m:n관계에서 한 표의 매 행들은 다른 표에서 많은 대응하는 행들을 가질수 있다. 관계형자료기지에서 m:n관계는 직접 모형화할수 없다. 따라서 그것들은 여러개의 1:n관계로 분해되여야 한다.

앞에서 본 주문항목표는 m:n관계를 여러개의 1:n관계로 분해하는 과정을 보여준다. 주문관계에서 주문된 상품들과 상품항목들은 m:n관계로 련관된다. 한 주문에서 여러개의 상품항목들을 주문할수도 있고 같은 상품이 여러 주문에 제기될수도 있다. 상품항목표는 상품항목과 주문사이의 1:n관계를 실현하는데 리용된다.

1.1.5. 뷰

자료기지에서 자료는 사용자에게 뷰(view)라고 부르는 여러가지 론리조합으로 현시될수 있다. 모든 뷰들은 표에 적용하는것과 같은 자료조작능력을 지원한다.

사용자들은 자료기지에서 자료를 선택하여 뷰(view)라고 하는 림시표들을 작성할수 있다. 이 뷰들은 일반적으로 그것을 작성할 때 리용된 선택항목에 따르는 이름으로 보관된다. 뷰에는 보통의 표들과 꼭 같은 방법으로 접근할수 있다.

뷰들은 보통 이미 존재하는 표에서 자료들의 부분모임을 작성하는데 리용된다. 표 1-6은 표 1-1에서 일부 행들만을 보여주는 실례이다.

표 1-6. 평양시 중구역의 주문자들에 대한 뷰

ID	Family_ Name(성)	Personal_ Name(이름)	Sex (성별)	State (도)	City (시)	District (구역/군)	Dong (동/리)	Neighbor (인민반)
100	김	성철	남	평양시	평양	중구역	교구동	32
104	강	권혁	남	평양시	평양	중구역	류성동	23

1.1.6. 정규화

정규화(Normalization)란 정규형으로 알려진 규칙들에 따라 자료기지에서 자료를 조직화하는 과정을 말한다. **정규형**(normal form)은 중복을 제한하고 일관한 의존관계를 담보하도록 하기 위한 자료기지설계의 지침이다. 자료가 중복되면 자료의 일치성을 보장할수 없다. 례를 들어 주문자의 주소를 두 장소에 보관하는 경우 주소가 변하면 그것이 두 장소에 동시에 반영되지 않을수도 있다. 그러므로 그 변화가 절대적으로 일치하도록 해야 한다.

사용자가 론리적이며 일관한 방법으로 자료에 접근하고 자료를 다룰수 있도록 자료의존성에 모순이 없다는것을 담보하는것이 중요하다.

정규화는 중복과 불일치를 최소화함으로써 완정성을 높여준다. 반면에 정규화규칙을 적용하면 자료가 여러 레코드들에 재분배될수 있으므로 자료검색효률이 떨어질수 있다.

정규화규칙에 따르는 자료기지를 **정규형자료기지**라고 한다. 자료기지가 첫번째 정규화규칙에 따르면 1차정규형이라고 하고 간단히 1NF로 표기한다. 자료기지가 세번째 정규화규칙에 따르면 3차정규형(3NF)이라고 한다.

1차정규형

1차정규형의 요구는 다음과 같다.

- 모든 레코드들은 같은 수의 마당을 가진다.
- 모든 마당들은 오직 한개의 자료항목만을 포함한다.
- 반복되는 마당이 없어야 한다.

첫번째 요구는 모든 레코드어커런스*들이 같은 수의 마당을 포함하여야 한다는것이다.

두번째 요구는 자료항목들을 개별적으로 탐색할수 있도록 하기 위한것으로서 원자성요구로 알려져있다. 매 자료항목이 레코드에서 하나의 마당에만 저장된다는 요구는 자료의 완정성을 담보하는데서도 중요하다.

끝으로 표에서 매개 행은 기본열쇠로 식별할수 있어야 한다.

2차정규형

2차정규형의 요구는 다음과 같다.

---

* 렬이름들만으로 구성된 레코드의 정의를 레코드형이라고 하며 실제 렬값들로 구성된 레코드를 레코드어커런스라고 한다. 보통 레코드라고 하면 레코드어커런스를 의미하지만 경우에 따라서는 레코드형을 의미할 때도 있으므로 문맥에 따라 구별해보아야 한다.

- 표는 1차정규형이여야 한다.
- 표는 열쇠전체(열쇠의 일부가 아니라 열쇠를 이루는 전체 렬)와 관련되지 않는 자료마당들을 포함하지 말아야 한다. 보다 엄밀하게는 기본열쇠에 포함되지 않는 모든 렬들이 기본열쇠에 완전함수종속**이여야 한다.

2차정규형은 표가 여러개 렬로 구성된 열쇠를 가질 때에만 적절하다. 례를 들어 매 창고에 있는 상품목록을 보여주는 표 1-7에서 행을 식별하는 유일한 수단인 기본열쇠는 두개의 렬 즉 품명마당과 창고(번호)이다.

2차정규형은 표가 오직 하나의 개체와 관련된 자료만을 포함해야 한다는것을 요구한다. 창고상품목록표는 주어진 창고의 상품항목들을 서술하려는것이며 따라서 상품항목 그 자체와 관련된 모든 자료는 기본열쇠와 관계된다.

표 1-7의 실례에서 두번째 행은 창고 #2에 백두산들쭉술이 97상자 있고 그것의 원가가 1.95라는것을 보여준다. 3번째 행은 창고 #7에 원가가 2.05인 백두산들쭉술이 103상자 있다는것을 말해준다. 이 표에서 창고주소는 열쇠의 일부분인 창고에 종속되며 따라서 전체열쇠에 완전종속이 아니다. 그러므로 창고주소마당은 2차정규형의 표에 속할수 없다. 만일 이 정보가 모든 상품항목들에 포함되면 명백히 정의된 주참조가 없기때문에 다른 행들에 있는 주어진 창고에 대한 주소들의 불일치를 피할수 없게 된다.

물론 이외에도 같은 자료항목을 여러 위치에 저장하면 기억공간이 랑비되고 자료항목이 변경될 때마다 단일한 주참조가 아니라 그 자료를 포함하는 모든 행들에 대하여 변화가 이루어지기때문에 자원리용면에서도 비효률적이다.

표 1-7. 창고상품목록표

Name (품명)	Warehouse (창고)	Address (주소)	Description (품종)	Qty (량)	Cost (원가)
고려인삼술	창고 #2	광복거리	주류	178	1.95
백두산들쭉술	창고 #2	광복거리	주류	97	1.95
백두산들쭉술	창고 #7	통일거리	주류	103	2.05
강계인풍술	창고 #7	통일거리	주류	15	0.98

** 어떤 표에서 X와 Y가 각각 렬모임의 부분모임이라고 할 때 X의 매개 값에 대하여 시간에 관계없이 Y의 값이 오직 하나만 련관된다면 Y는 X에 함수종속이라고 한다. 어떤 렬 y가 다른 렬모임 X(례컨대 복합열쇠)에 함수종속이고 그의 부분모임(복합열쇠를 이루는 개별적인 렬이나 렬모임)에는 함수종속이 아닐 때 y는 X에 완전함수종속이라고 한다.

이 문제의 해결책은 창고주소를 외부열쇠에 의해 창고상품목록표와 련관되는 창고표에 옮기는것이다. 결과적인 2차정규형의 표들을 표 1-8과 표 1-9에 보여주었다.

표 1-8. 2NF의 창고상품목록표

Name (품명)	Warehouse (창고)	Description (품종)	Qty (량)	Cost (원가)
고려인삼술	창고 #2	주류	178	1.95
백두산들쭉술	창고 #2	주류	97	1.95
백두산들쭉술	창고 #7	주류	103	2.05
강계인풍술	창고 #7	주류	15	0.98

표 1-9. 2NF의 창고표

Warehouse(창고)	Address(주소)
창고 #2	광복거리
창고 #7	통일거리

개괄적으로 말하면 2차정규형은 전체 열쇠와 직접적으로 관련되지 않는 자료는 제거하여 분리된 다른 표에 배치할것을 요구한다. 이 새로운 표들은 외부열쇠를 리용하여 본래의 표와 련결되여야 한다. 표 1-8과 표 1-9의 실례에서 창고렬은 표 1-8에서 기본열쇠의 일부이며 표 1-9를 지적하는 외부열쇠이다.

3차정규형

3차정규형의 요구는 다음과 같다.

- 표는 2차정규형으로 되여야 한다.
- 표는 기본열쇠와 관련되지 않는 마당들을 포함할수 없다. 보다 엄밀하게는 기본열쇠에 속하지 않는 모든 렬들이 기본열쇠에 이행적함수종속*이 아니여야 한다.

3차정규형은 복합열쇠가 아니라 하나의 열쇠를 포함하는 경우를 론한다는것을 제외하면 2차정규형과 대단히 류사하다. 2차정규형의 설명에 리용된 실례에서는 《백두산들쭉술》과 같이 같은 이름의 상품항목이 창고번호와 같이 속성이 각이할수 있기때문에 복합열쇠가 리용되였다. 표 1-10에 보여준 종업원표는 하나의 열쇠를 가지지만 2차정규형의 실례와 류사한 경우이다.

* 일반적으로 함수종속관계 A→B와 B→C가 성립되면 론리적 결과로 A→C가 성립된다. 이때 렬 C는 렬A에 이행적함수종속이라고 한다.

표 1-10. 종업원표(Employee)

Name(이름)	Department(부서)	Location(거주지)
김영철	판매과	문수거리
리성호	생산과	버드나무거리
오수영	운수과	광복거리

표 1-10에서 거주지렬은 부서의 지역적위치를 설명하고있다. 모든 종업원은 반드시 어떤 부서에 속해있으며 그 부서가 위치하는 곳에서 일하게 된다. 3차정규형의 요구에 만족하자면 기본열쇠를 서술하는 자료를 포함하지 않는 렬은 다른 표에 옮겨져야 한다. 이 경우 부서의 거주지는 기본열쇠인 종업원이름과 직접 련관된 자료는 아니고 부서를 통하여 이행적인 종속관계에 놓인다. 따라서 종업원표가 3차정규형에 따르기 위해서는 종업원이름과 부서이름, 부서이름과 부서의 위치를 포함하는 두개의 표로 분리되여야 한다. 이때 종업원표에서 부서표를 가리키는 외부열쇠는 부서렬이다. 결과적인 표들을 표 1-11과 표 1-12에 보여주었다.

표 1-11. 정규화된 종업원표

Name	Department
김영철	판매과
리성호	생산과
오정식	운수과

표 1-12. 부서표(Departments)

Department	Location
판매과	문수거리
생산과	버드나무거리
운수과	광복거리

4차정규형

4차정규형의 요구는 다음과 같다.

- 표는 3차정규형이여야 한다.
- 표는 어떤 개체에 대한 2개 이상의 독립적인 다값요소들을 포함할수 없다.

례를 들어 주문자들의 전화번호를 보관하려면 주문자ID렬, 전화(보통의 유선전화)번호렬, 팍스번호렬, 손전화번호렬을 포함하는 표를 만들어야 할것이다. 주문자가 표에 렬거된 매개 렬(속성)들에 대하여 하나의 번호만을 가진다면 문제가 없다. 그러나 주문자가 유선전화번호 2개, 팍스번호 1개, 손전화번호 2개를 가진다면 주문자 ID(Customer_ID)를 외부열쇠로 하는 표 1-13과 같은 표를 만들게 될것이다.

표 1-13. 4NF에 위반되는 전화번호표

CUSTOMER_ID	PHONE	FAX	CELL
100	123-234-3456	123-234-3460	121-345-5678
100	123-234-3457	<NULL>	121-345-5679

이 표에서 각이한 전화번호렬들은 서로 독립이며 매 행이 주문자에 대하여 두개(유선전화번호렬과 손전화번호렬)의 독립적인 다값요소를 포함하고있다. 따라서 이 표는 4차정규형에 위반된다. 주어진 행에서 유선전화, 팍스, 손전화번호의 조합은 의미가 없다는데 주의하기 바란다.

4차정규형의 위반과 관련한 기본문제는 자료를 유지하기 위한 명백한 방법이 없다는것이다. 례를 들어 만일 주문자가 첫 행에 렬거된 손전화를 없애려는 경우 두번째 행의 전화번호를 첫번째 행에로 옮겨야 하는가 아니면 거기에 그냥 남겨두어야 하는가? 또 첫 행의 손전화번호와 두번째 행의 유선전화번호를 포기하는 경우에는 모든 전화번호들을 한개 행으로 통합정리해야 하는가? 분명히 이 자료기지의 유지는 대단히 복잡해진다.

문제의 해결은 본래의 표로부터 전화, 팍스, 손전화렬을 없애고 외부열쇠로 주문자 ID를 가지고 자료마당으로는 전화번호와 그의 류형을 포함하도록 하는것에 의해 이 문제를 우회하는것이다.(표 1-14) 이것은 4차정규형을 위반함이 없이 매 주문자에 대하여 다른 류형의 여러가지 전화번호들을 다룰수 있게 한다.

표 1-14. 전화번호표(Phone Numbers Table)

CUSTOMER_ID	NUMBER	TYPE
100	123-234-3456	PHONE
100	123-234-3457	PHONE
100	123-234-3460	FAX
100	121-345-5678	CELL
100	121-345-5679	CELL

5차정규형

5차정규형의 요구는 다음과 같다.

- 표는 4차정규형이여야 한다.
- 표를 론리적으로 다른 기본열쇠를 가지는 보다 작은 표들로 분해할수 없어야 한다.

5차정규형은 4차정규형과 류사한데 4차정규형이 독립적인 다값요소들을 다룬다면 5차정규형은 호상의존하는 다값요소들을 다룬다. 례를 들어 각이한 공장들로부터 들어온 여러가지 류사한 제품계렬들을 취급하는 판매소를 생각해보자. 제품을 팔기전에 판매자는 그 제품에 익숙되여야 한다. 표 1-15에 그러한 상태를 보여주었다.

표 1-15. 판매원표(Salespersons)

Salesperson (판매자)	Factory (공장)	Product (제품명)
라영호	대안전기	세탁기
라영호	대안전기	랭동기
라영호	보통강전기	세탁기
라영호	보통강전기	랭동기

이 표는 일정한 중복을 포함하고있다. 표에서 공장과 제품명은 판매자에 대하여 호상의존하는 다값요소들이다. 이 중복은 표를 5차정규형으로 변환하여 제거할수 있다. 5차정규형으로의 변환은 표 1-16부터 표 1-18까지와 같이 표를 보다 작은 표들로 분해하여 실현할수 있다.

표 1-16. 공장별 판매자(Salespersons by Factory)

판매자	공장
라영호	대안전기
라영호	보통강전기

표 1-17. 제품별 판매자(Salespersons by Product)

판매자	제품
라영호	세탁기
라영호	랭동기

표 1-18. 공장별 제품(Products by Vendor)

공장	제품
대안전기	세탁기
대안전기	랭동기
보통강전기	세탁기
보통강전기	랭동기

보이스-코드(Boyce-Codd)정규형

보이스-코드정규형(BCNF)은 3차정규형의 보다 엄밀한 판본으로서 다음과 같은 항목들을 포함하는 자료를 다루기 위한것이다.

- 여러개의 후보열쇠
- 복합후보열쇠
- 중첩된 후보열쇠

관계표는 모든 렬들이 후보열쇠이고 그중 일부 렬들이 함수적으로 완전히 종속되여 있는 경우에만 BCNF로 된다. 다시말하여 표가 기본열쇠(후보열쇠라고 부르는)로 리용될수 있는 여러개의 렬 혹은 렬들의 묶음을 가지는 경우 그 표가 BCNF이기 위해서는 이 매 후보열쇠들에 대하여 3차정규형이여야 한다.

실천에서의 정규화

대부분의 자료기지는 5차정규형일 때 충분히 정규화된것으로 볼수 있다. 5차정규형에서 자료기지는 다음과 같은 중요한 성질들을 가진다.

- 모든 레코드들은 같은 수의 마당을 가진다.
- 모든 자료마당들은 단일한 자료항목을 포함한다.
- 반복되는 마당이 없다.
- 모든 마당들은 전체기본열쇠와 관련이 없는 자료는 포함하지 않는다.
- 표는 열쇠에 관하여 둘 이상의 독립적인 다값요소를 포함하지 않는다.
- 표는 열쇠에 관하여 둘 이상의 호상의존하는 다값요소를 포함하지 않는다.

추가적인 정규형은 특수한 경우에만 적용된다. 레를 들어 BCNF는 기본열쇠로 리용할수 있는 속성을 가진 모든 렬 혹은 렬들의 묶음에 대하여 표가 3차정규형일것을 요구한다. 다시말하여 모든 후보열쇠들에 대하여 표가 3차정규형일것을 요구한다. 그러나 실천적으로 자료기지설계자들은 기본열쇠를 결정하면 다른 후보열쇠들은 리용하지 않으며 결국은 3차정규형이면 충분하다.

제2절. 고수준언어

자료기지관리체계의 고수준언어들은 자료기지리용을 위한 언어의 리용가능성과 관련된다. 그것들은 선형적인 문장구성법을 가져야 하며 다음의 기능들을 지원하여야 한다.

- 자료정의(뷰정의 포함)
- 자료의 갱신과 검색
- 자료-완정성제약조건
- 거래관리
- 자료보안제약조건

이 기능들을 모두 표준적으로 실현하고있는 언어가 바로 **구조화질문언어**(SQL: Structured Query Language)이다. SQL은 다음과 같은 부분언어들의 합성으로 이루어져있다.

- 자료정의언어(DDL) - 표와 색인들을 생성, 변경, 제거하는데 리용된다.
- 자료조작언어(DML) - 자료를 삽입, 갱신, 삭제하는데 리용된다.
- 자료질문언어(DQL) - SELECT지령을 리용하여 자료기지에 질문하는데 리용된다.
- 거래조종지령 - 거래의 시작, 위탁 혹은 되돌리기(혹은 취소)하는데 리용된다.
- 자료조종언어(DCL) - 사용자특권의 허가와 취소, 통과암호의 변화에 리용된다.

SQL이 여러개의 부분언어들로 분할되여있지만 임의의 부분언어들에 속한 문들이 함께 리용될수 있다. 아래에 자료기지의 기본적인 기능을 수행하는데 리용되는 부분언어들을 소개한다.

1.2.1. 자료정의언어

자료정의언어(Data Definition Language)는 자료기지의 생성과 변경에 리용된다. DDL의 기본지령들은 CREATE, ALTER, DROP이다.

DDL리용의 좋은 실례는 표생성이다. 표가 생성될 때 매개 렬에 여러가지 파라메터들이 설정된다. CREATE TABLE지령의 기본형식은 다음과 같다.

```
CREATE TABLE 표이름
(렬이름 자료형 [(크기)] [제약] [기정값], ...);
```

- 자료형 - 여기에는 문자형, 옹근수형, 류점수형 등 기타 자료형들이 포함된다.
- 자료제약조건 - NULL값이 허용되는것과 같은 제한조건들이 포함된다.
- 기정값 - 기정값들은 매개 렬에 할당될수 있다.

완정성제약과 트리거

기본열쇠와 외부열쇠에 대하여 고찰한 앞의 내용들을 참고해보면 기본열쇠가 NULL 값을 가지거나 혹은 유일하지 않은 값을 가지는 경우 열쇠들을 리용하여 표들을 련관시키는것이 곤난하다는것을 알수 있다. 이와 같은 문제들은 제약(constraint) 혹은 제약조건으로 해결한다. 제약의 기본류형은 다음과 같다.

- NULL 혹은 NOT NULL제약은 마당이 꼭 유효한 값을 가져야 하는가 아니면 마당을 비워둘수 있는가를 규정한다.
- UNIQUE제약은 어떤 특정한 렬에 대하여 같은 값을 가지는 레코드어커런스가 있을수 없다는것을 규정한다.
- PRIMARY KEY제약은 이 렬이 표에 대한 기본열쇠라는것을 지적한다.

SQL언어는 제약들을 정의할 뿐만아니라 사용자가 어떤 표에 지적된 연산들이 수행될 때 적용되는 보안규칙들을 규정할수 있게 한다. 이 보안규칙들을 **트리거**(trigger)라고 부르며 이름에 의해 호출되는것이 아니라 표를 갱신하는것과 같은 자료기지사건이 일어날 때 자동적으로 발생한다는것이 좀 다를뿐 저장수속과 꼭같이 동작한다.

트리거를 리용하는 전형적인 실례로 상품목록표에서 진행한 갱신의 유효성검사를 들수 있다. 다음의 코드는 상품목록표에서 상품의 가격을 15%이상 증가시키려는 경우 자동적으로 되돌리기 혹은 취소하는 트리거를 보여준다.

```
CREATE TRIGGER FifteenPctRule ON INVENTORY FOR INSERT, UPDATE AS
DECLARE @NewCost money
DECLARE @OldCost money
SELECT @NewCost = cost FROM Inserted
SELECT @OldCost = cost FROM Deleted
IF @NewCost > (@OldCost * 1.15)
ROLLBACK Transaction;
```

거래관리와 SQL의 ROLLBACK지령에 대해서는 뒤에서 인차 학습하게 된다.

1.2.2. 자료조작언어

자료조작언어는 표에 자료를 삽입하거나 표의 내용을 갱신 및 삭제하는데 리용되는 3가지 SQL문들로 구성되여있다.

- INSERT
- UPDATE
- DELETE

INSERT문은 한번에 한개 행(레코드)단위로 자료를 삽입하는데 리용된다. 또한 SELECT문과 결합하여 다른 표들로부터 하나이상의 행들을 선택하여 삽입하는데 리용할수 있다.

UPDATE문은 행에서 개별적인 렬들의 내용을 변경하는데 리용된다. UPDATE문은 보통 갱신할 행들을 선택하기 위하여 WHERE문절과 함께 리용된다.

DELETE문은 표에서 선택된 행을 삭제하는데 리용된다. 여기서의 행선택 역시 WHERE문절의 결과에 기초한다.

1.2.3. 자료질문언어

자료질문언어는 자료기지에서 질문에 해당한 응답자료를 추출하는데 리용하는 SQL 부분언어이다. SELECT문은 자료질문언어의 심장이다. SELECT문은 질문에 대한 자료추출외에도 특정한 레코드들을 변경하는것과 같은 다양한 조작들에서 자료를 선택하기 위하여 다른 SQL언어들과 결합하여 리용할수 있다.

그러나 SELECT문의 가장 일반적인 리용은 자료기지에 대한 자료검색지령 혹은 자료검색질문이다. SELECT문의 기본형식은 다음과 같다.

```
SELECT 렬이름1, 렬이름2, ... FROM 표이름;
```

우의 형식은 질문에 필요한 마당이름들을 모두 렬거하는 형식이다.

SQL에서는 SELECT문에서 대용기호형식도 지원한다. 대용기호형식에서 별표(*)는 모든 렬을 대신하게 된다. 별표는 표의 모든 렬들에 대한 값을 요구할 때 리용된다.

```
SELECT * FROM 표이름;
```

SELECT지령의 실제적인 위력은 WHERE문절과 함께 리용될 때 나타난다. WHERE문절은 사용자가 어떤 일정한 기준에 맞는 레코드들에 한해서 요구된 마당들만 돌려주도록 질문을 제한한다. 례를 들어 표 1-1에 보여준 주문자표에 다음과 같은 질문을 할수 있다.

```
SELECT * FROM Customers WHERE Family_Name = '김'
```

이 질문이 수행되면 《김》가성을 가진 주문자들에 대한 모든 렬들을 되돌린다. 되돌려지는 렬들의 순서는 그것들이 자료기지에 저장된 순서로서 임의적이다.

질문결과의 정렬

SELECT문을 리용하여 관계형자료기지관리체계로부터 자료를 추출할 때 공통적으로 제기되는 요구는 자모순 혹은 수값순서로 질문결과들을 정렬하는것이다. 결과에 대한 정렬은 ORDER BY문절을 리용한다.

```
SELECT Family_Name, Personal_Name, State, City
FROM Customers
WHERE Family_Name = '김'
ORDER BY Personal_Name;
```

표들의 결합

자료기지에서 정보는 보통 론리적으로 련관된 자료모임이 들어있는 여러개의 표들에 분산보관된다. 그러한 자료기지의 실례를 표 1-1과 표 1-3부터 1-5에 주었다.

주문자가 어떤 상품을 주문하면 주문번호가 할당되고 주문자ID와 주문날자를 포함하는 항목이 주문표(Orders)에 만들어진다. 다음에는 주문항목표(Ordered_Items)에 주문번호, 품목번호, 량을 기록하는 항목들이 추가된다.

SQL의 가장 강력한 기능의 하나가 JOIN을 리용하여 여러 표들의 자료를 결합하는 능력이다. 실례로 다음의 SQL지령은 주문자들에게 판매된 판매총액을 얻기 위하여 ORDERS, CUSTOMERS, ORDERED_ITEMS, INVENTORY표들에 JOIN을 실시하고있다.

```
SELECT c.Family_Name + c.Personal_Name AS Name,
   SUM(oi.Qty * Cost * 1.6) AS PURCHASES
FROM Orders o, Customers c, Ordered_Items oi, Inventory i
WHERE o.Customer_Id = c.Customer_Id AND
   o.Order_Number = oi.Order_Number AND
   oi.Item_Number = i.Item_Number
GROUP BY c.Family_Name + c.Personal_Name;
```

이 질문의 결과는 다음과 같다.

Name	Purchases
강권혁	21.52
김성철	2.02
김　철	2.59
김은희	11.87

이 실례에서는 산수연산과 문자렬연산을 수행하는 SQL의 능력을 보여주는것과 함께 렬이름과 표에 대한 별명을 리용하는 법을 설명해주고있다. 여기서 별명 NAME은 《성》마당과 《이름》마당을 련결하여 하나의 마당으로 보여줄 때 렬이름에 할당되며 별명 PURCHASES에는 수량과 원가를 곱하고 거기에 판매부과률을 곱한 값이 할당되였다.

보고서작성기능

SQL은 자료요소들에 대한 통계적 및 포괄적인 정보를 제공하는데 리용할수 있는 많은 **집약**(aggregation)기능들을 지원한다. 표준적인 집약기능에는 다음과 같은것들이 포함된다.

- 합과 계수
- 평균값과 표준편차
- 최대값과 최소값

간단한 판매보고서는 집약기능을 리용하는 좋은 실례이다. 다음의 질문은 지역별로 판매된 상품의 총비용(원가)과 판매액을 렬거한 결과모임을 작성한다.

```
SELECT STATE, SUM(oi.QTY * COST) AS TOTAL,
   SUM(oi.QTY * COST * 1.6) AS SALES
FROM Orders o, Customers c, Ordered_Items oi, Inventory i
WHERE o.Customer_ID = c.Customer_ID AND
   o.Order_Number = oi.Order_Number AND
   oi.Item_Number = i.Item_Number
GROUP BY State;
```

결과표는 다음과 같다.

STATE	TOTAL	SALES
함경남도	1.62	2.592
평양시	22.13	35.408

지금까지 자료기지와 표의 작성, 자료입력 및 검색에 대하여 간단히 학습하였다. 간단한 자료기지를 조작하는데는 지금까지 배운 내용을 가지고도 충분하지만 실천에서 제기되는 문제들은 보다 복잡하다. 실천에서는 크게 다음의 두 측면에서 문제가 발생한다.

- 많은 실천적인 응용프로그람들에서 완전한 조작은 간단한 하나의 SQL지령만으로 표현할수 없다. 따라서 여러개의 호상의존하는 문들을 조종하는 수단이 필요하다.
- 특히 보다 큰 체계들에서는 응용프로그람의 보안을 담보하는 어떤 수단이 제공되여야 한다.

이 요구들은 각각 거래조종지령들과 자료조종언어에 의해 처리된다.

1.2.4. 거래관리지령과 거래조종지령

거래관리(transaction management)는 거래라는 그룹화된 자료기지지령들을 실행하기 위한 관계형자료기지체계의 능력에 의거한다. **거래**(transaction)는 순서대로 수행되여야 하며 완전히 성과적으로 완성되여야 하는 지령들의 묶음 혹은 렬이다. 거래조종지령들은 거래를 조종하는데 리용된다.

ACID검사

자료처리에서 일반적으로 사용되는 표현법은 자료기지관리체계가 거래를 다루는데 필요한 속성들의 모임을 정의하는 ACID검사이다. 그 속성들은 다음과 같다.

- 원자성(Atomicity)
- 일관성(Consistency)
- 격리(Isolation)
- 영구성(Durability)

거래는 **원자성**을 가져야 한다. 즉 거래를 구성하는 모든 조작들은 불가분이며 모든 변화들이 효력을 발생하도록 전체가 위탁되든가 아니면 그것들중 어느것도 효력을 발생하지 않도록 전체가 본래상태로 되돌아가야 한다. 원자적인 거래의 전형적인 실례는 당좌예금으로부터 저축예금에로 자금의 이동이다. 명백히 당좌예금으로부터의 공제와 저축예금에로의 추가가 둘다 일어나든가 그렇지 않으면 어느 한쪽도 일어나지 말아야 할것이다. 원자성이 담보되지 않는 경우에는 어느 한쪽에서만 변화가 일어나(실례로 당좌예금에서 공제된 금액이 저축예금에 나타나지 않거나 저축예금에 추가된 금액이 당좌예금에서는 공제되지 않는 현상) 예금자와 은행의 어느 한쪽이 손해를 보게 될것이다.

거래는 **일치성**요구를 만족시켜야 한다. 일치성요구는 거래가 사용자가 정의한 완정성제약에 따를 때에만 그것을 정당한것으로 정의한다는것이다. 본질적으로 이 제약들은 자료기지상태들을 정의하고 어떤 비정상적인 상태를 야기시킬수 있는 거래들을 금지시킨

다. 례를 들어 당좌예금으로부터 저축예금으로 자금을 옮기고 해당 업무규칙들이 그러한 자금이동을 다른 별개의 표에 기록할것을 요구한다면 그 표를 갱신하는 임의의 문제들은 완정성제약조건에 위반될것이며 전체 거래에 되돌리기가 요구될것이다.

거래는 현재의 거래가 끝날 때까지 그 효과가 다른 거래에 공개되지 말아야 한다. 이것을 거래의 **격리**라고 한다. 례를 들어 당좌예금에서 저축예금에로 자금을 이동하는 경우 저축예금이 차방(계산자리 왼쪽)에 기입된 다음 확인이 대방(계산자리 오른쪽)에 기입될 때까지 중간 차액잔고가 외부의 거래에 리용되지 말아야 한다. 만일 도중 차액잔고가 외부의 거래에 리용되면 례컨데 자금이 어느 계산자리에도 나타나지 않아 자금부족 경고가 발생할수 있다.

거래의 **영구성**은 거래가 일단 위탁되면 그 결과를 《영구적》인 저장장치에 보존할것을 요구한다. 다시말하여 당좌예금에서 저축예금에로 예금을 이동한 다음에는 자료기지관리체계가 그것을 영구저장장치에 보존하여야 한다.

SQL에서 거래관리

거래처리기간에 어떤 좋지 못한 일이 생기면 자료기지관리체계는 전체 거래를 무효 즉 되돌리기한다. 한편 거래가 성과적으로 완료되면 그것을 자료기지에 보관 즉 위탁한다.

거래는 보통 우에서 본 은행예금이동에서처럼 여러개의 련관지령들을 포함한다. 만일 의뢰자가 자기의 당좌예금으로부터 저축예금에로 자금을 옮기는 경우 적어도 다음 두개의 자료기지접근지령이 실행되여야 한다.

- 저축예금이 차방(계산자리왼쪽)에 기입되여야 한다.
- 당좌예금이 대방(계산자리오른쪽) 기입되여야 한다.

만일 이 지령들중 하나가 수행되고 다른것은 수행되지 않는 경우 금액은 저축예금에 나타나지 않고 당좌예금에서만 없어지든지 당좌예금에서 공제하지 못한 상태에서 저축예금에 나타날것이다.

이 문제를 해결하기 위한 방도는 론리적으로 련관된 지령들을 단일한 거래로 위탁되는 묶음으로 결합하는것이다. 만일 어떤 문제가 생기면 거래전체가 거래전 상태로 되돌리기때문에 업무전반에 악영향을 주지 않고 그 문제를 고착시킬수 있다.

SQL에서는 COMMIT지령과 ROLLBACK지령들을 통하여 이 요구를 지원한다. COMMIT지령은 거래가 시작된 때로부터 그 지령이 발생된 시점까지 일어난 변화들을 위탁하며 ROLLBACK지령은 그것들을 본래상태로 되돌려보낸다.

대부분의 자료기지들은 모든 지령들을 개별적으로 실행되는 차제로 자료기지관리체계에 위탁하는 AUTOCOMMIT를 지원한다. 이 추가선택은 SET지령으로 설정 혹은 해제할수 있다. 기정으로는 AUTOCOMMIT가 설정상태이다.

1.2.5. 자료기지보안과 자료조종언어

자료기지는 일반적으로 많은 시간과 로력이 투입된것이며 기관의 자산으로 되기때문에 보안을 담보하는것이 매우 중요하다. 자료기지보안의 가장 중요한 특징들은 다음과 같다.

- 자료기지접근은 일반적으로 통과암호원리의 확장에 의해 위임된 자격있는 사람에게만 허용한다.
- 많은 사용자들이 동시에 접근 및 갱신하는 자료기지의 일관성을 담보한다.
- 자료기지의 물리적인 완정성을 담보한다. 최근에 이것은 파일대피와 재적재를 위한 저장을 포함한다.

대부분의 자료기지관리체계들은 자료기지보안을 관리하기 위한 전용도구들을 내장하고있다. 일반적으로 접근조종기구들은 서로 비슷하며 SQL언어를 리용한다.

자료기지사용자에 대한 관리

자료기지용어로 말하면 **사용자**는 자료기지에 접근하는 임의의 사람을 말한다. 대부분의 자료기지관리체계에서는 자료기지보안을 관리하기 위한 각이한 접근특권과 각이한 조작임무를 지닌 서로 다른 사용자 혹은 사용자그룹을 정의하는 능력을 제공한다. 자료기지가 작성될 때 그 작성자는 소유자특권을 가진다. 소유자특권을 가진 사용자는 자료기지와 그 안의 임의의 구성요소들을 창조할수 있다. 자료기지가 작성된 다음에는 소유자보다 낮은 특권을 가진 사용자들이 자료기지에 접근할수 있다. 실례로 자료를 입력하는 사무원은 지적된 표들에 제한된 자료를 입력하는데 필요한 특권만을 가진다.

자료기지관리자가 자료기지에 접근할수 있는 개별적인 사용자를 추가하려면 자료기지사용자를 창조하여야 한다. 사용자의 추가는 CREATE USER지령으로 진행한다.

자료기지사용자의 존속기간에 사용자의 통과암호나 접근만기날자를 변경시키려면 ALTER USER지령을 리용한다.

극단적으로 어떤 사용자의 자료기지전체에 대한 접근을 완전히 취소시키려는 경우 DROP USER지령을 리용한다.

사용자특권

자료기지관리체계는 사용자들에게 자료기지에 대한 여러가지 특권들을 할당해준다. 이 특권들은 자료기지의 객체들에 대하여 수행할수 있는 작용들에 대응된다. 이 방법은 자료기지접근조종의 등급을 제공한다. 즉 자료기지관리자에게는 요구하는 모든것을 다 할수 있게 하지만 일반 사무원들은 보다 낮은 특권을 주어 나타날수 있는 실수나 위험을 줄일수 있게 한다.

새로운 자료기지를 작성할 때 자료기지의 기정소유자는 CREATE지령을 실행시키

는 사용자이다. 다른 사용자들이 자료기지와 작업하도록 허용하기 위해서는 해당한 특권을 그들에게 할당해야 한다. 특권은 개별적인 사용자 혹은 사용자그룹에 할당될수 있다.

사용자그룹과 역할

많은 체계들에서는 자료기지관리자가 개별적인 사용자들을 정의할뿐만아니라 사용자들을 같은 특권을 가진 론리적인 그룹으로 편성할수 있게 한다. 사용자그룹과 그의 역할도 개별적인 사용자와 마찬가지로 SQL지령을 리용하여 관리한다.

역할은《자료기지의 창조》,《자료기지 여벌복사》등과 같이 자료기지에 대하여 사용자들이 할수 있는 조작들을 정의한다. 다시말하여 역할은 미리 정의된 사용자특권들의 모임이다.

사용자그룹의 창조, 변경 및 제거는 개별적인 사용자를 창조할 때와 거의 같은 방법으로 진행할수 있다. 그룹들은 또한 개별적인 사용자들이 다른 그룹에 속할수 있듯이 또 다른 그룹에 속할수 있다.

그룹이 변경 및 제거될 때에는 오직 그 그룹만이 영향을 받는다. 제거되는 그룹에 속한 임의의 사용자는 단순히 그 그룹성원으로서의 자격만 잃을뿐 그 밖의 다른 영향은 받지 않는다. 류사하게 사용자제거로 하여 어떤 그룹이 변경될 때에도 그 그룹만이 영향을 받는다. 사용자는 단순히 그 그룹의 성원자격만 잃을뿐 다른 영향은 받지 않는다.

사용자역할들은 단순히 미리 정의된 사용자특권들의 모임이다. 사용자역할은 자료기지관리자가 시간을 절약할수 있도록 설계된 순수 관리적인 기능이다. 그룹과 마찬가지로 그룹의 역할도 필요에 따라 자료기지관리자가 정의할수 있다.

제3절. 자료기지구성방식

관계형자료기지관리체계는 현재의 응용프로그람들이 자료기지관리체계의 분산된 판본들을 도입할 때 혹은 현재 분산된 자료가 체계주위에 재분산될 때에도 성공적으로 동작할수 있다는것을 담보한다. 분산된 체계에 대한 요구는 이미 콤퓨터활용의 초기에 제기되였다.

현대적인 체계들에서 분산은 여러가지 방식으로 실현된다. 코드가 말한 분산의 류형은 지금 관계형자료기지관리체계의 본질로 고찰되여야 하며 따라서 자료기지는 상당한 크기의 체계무리로 분산될수 있다. 그렇지만 아직 그의 분산은 그것을 단일한 관계형자료기지관리체계로 접근하려는 자바자료기지프로그람작성자에게는 투명한것으로 되여야 한다. 자바자료기지프로그람을 작성하는 프로그람작성자에게는 여러층 구성방식이 훨씬 더 일반적인 분산형식이다.

자료기지응용에서 가장 일반적으로 리용되고있는 체계구성방식은 2층모형과 3층모형이다. 다시말하여 자바응용프로그람이 자료기지에 직접 접근하든가 중간층 봉사기응용프로그람을 통하여 접근할수 있다. **단층**(single tier)으로 부를수도 있는 새로운 변종은 자바자료객체(Java Data Objects: JDO)에 기초하고있는 응용프로그람이다. 여기서 JDO는 자바프로그람작성자가 작성하는 특별한 지속(persistence)코드가 없이도 지속성을 보장해준다.

1.3.1. 자바자료객체

다른 객체지향언어에서와 마찬가지로 자바에서도 프로그람작성자는 기본적으로 객체와 작업하게 된다. 한편 관계형자료기지는 객체의 속성이라고 볼수 있는 보다 작은 자료항목들을 위주로 구성된다. 실례로 자바에서 주문자객체를 만드는 경우 이 객체는 이름과 성별, 거주지와 같은 속성들을 가지는데 이 속성들은 자료기지레코드에서 개별적인 렬들에 각각 저장된다.

JDO구성방식은 기초적인 지속성기구의 세부를 응용프로그람이 볼수 없게 숨겨주는 투명한 지속성개념을 지원한다. 자바업무론리는 관습적인 방식으로 단순하게 개발되였다. 업무론리클라스들은 지속가능한 클라스의 판본을 생성하기 위하여 바이트코드준위에서 강화된다. 거의 모든 사용자정의 클라스들은 이러한 방식으로 지속화될수 있다.

일단 업무클라스들이 콤파일되고 강화되면 강화된 업무클라스를 리용하는 응용프로그람을 개발할수 있다. 업무객체의 지속성관리는 투명하다. 즉 응용프로그람개발자는 JDBC/SQL준위에서 객체나 그의 속성들을 꺼내고 저장할 필요가 없다.

주의: 비록 JDO응용프로그람이 단층모형처럼 보이고 또 단층모형으로 동작하지만 밑준위에 있는 지속성수법은 국부관계형자료기지관리체계 혹은 구성방식이 기초하고있는 다층 EJB를 리용하여 수행된다.

1.3.2. 2층모형

2층모형에서 자바응용프로그람은 자료기지와 직접 호상작용하도록 설계된다. 응용프로그람의 기능은 다음의 두개 층으로 분할된다.

- 사용자대면부, 업무론리, JDBC구동프로그람을 포함하는 응용층
- 관계형자료기지관리체계를 포함하는 자료기지층

자료기지에 대한 대면은 구체적인 자료기지관리체계에 합당한 자바자료기지접속(Java Database Connectivity:JDBC)구동프로그람에 의해 조종된다. JDBC구동프로그람은 자료기지에 SQL문을 넘겨주고 그 문의 결과를 응용프로그람에 돌려준다.

그림 1-2에 보여준것과 같은 의뢰기/봉사기구성은 2층모형의 특수한 경우이다. 거기서 자료기지는 봉사기라고 부르는 다른 기계에 자리잡고있다. 응용프로그람은 망을 통하여 봉사기와 접속된 의뢰기에서 실행된다.

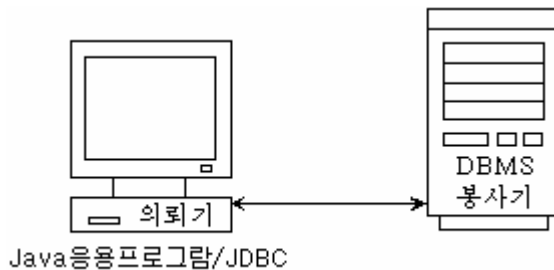



그림 1-2. 2층 의뢰기/봉사기구조

2장에서 2층응용프로그람의 내용을 이루는 JDBC와 SQL의 기초적인 기능들에 대하여 학습하게 된다. 응용프로그람들은 관계형자료기지체계의 도형사용자대면부(GUI)를 작성하기 위하여 단순한 Swing구성요소들을 리용한다. 자료기지응용프로그람개발에서 Java/JDBC방법을 리용하면 사용자들이 Oracle, Sybase, SQL Server, MySQL을 포함한 넓은 범위의 관계형자료기지체계들에 쉽게 접근할수 있다.

1.3.3. 3층모형

3층모형에서 의뢰기는 중간층을 이루는 응용프로그람봉사기에 요청을 보낸다. 그러면 응용프로그람봉사기가 그 요청들을 해석하고 그 수행에 필요한 SQL지령들을 형식화하여 자료기지에 보낸다. 자료기지는 이 SQL지령들을 처리하고 결과를 다시 응용프로그람봉사기에 보낸다. 응용프로그람봉사기는 그 결과를 받아서 의뢰기에 보내준다.

3층구성방식의 우점은 다음과 같다.

- 응용프로그람봉사기와 자료기지봉사기를 분리함으로써 성능을 보다 개선할수 있다.
- 업무론리가 자료기지로부터 완전히 분리된다.
- 의뢰기응용프로그람들은 CGI와 같은 간단한 규약을 리용하여 봉사기에 접근할수 있다.

그림 1-3에서 보여준 3층모형은 웨브응용프로그람들에서 보편적이다. 이 방식에서 의뢰기층은 보통 의뢰기의 열람프로그람이고 중간층은 써블레트엔진과 함께 웨브봉사기에서 실현되며 자료기지관리체계는 전용의 자료기지봉사기에서 수행된다.

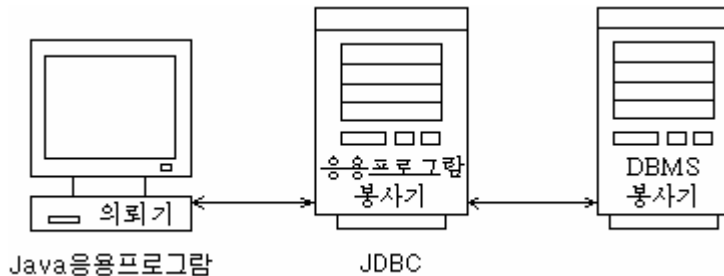



그림 1-3. 전형적인 웨브응용의 3층모형

3층구성방식의 주요구성요소는 다음과 같다.

- 의뢰기층(Client tier) - 이 층은 보통 웨브열람기를 리용하여 실현할수 있는 얇은 표현층이다.
- 중간층(Middle tier) - 이 층에서는 업무론리 혹은 응용론리를 처리한다. 이 층은 Tomcat와 같은 써블레트엔진이나 JBOSS와 같은 응용프로그람봉사기를 리용하여 실현된다.
- 자료원천층(Data source layer) - 여기에는 관계형자료기지관리체계가 포함된다.

5장에서 JDBC API의 추가적인 능력들을 학습하게 된다.

제4절. 자료기지의 설계

1.4.1. 자료기지설계의 고찰

지금까지 우리는 관계형자료기지가 어떤것인가에 대하여 보았다. 이 절에서는 실례 자료기지의 설계과정을 통하여 관계형자료기지의 설계방법과 메쏘드에 대하여 알아보기로 한다.

일반적으로 작은 창고같은것을 만들 때에는 간단한 설계로 충분하거나 특별한 설계과정이 없이 설계와 시공을 동시에 할수 있다. 그러나 큰 건물을 세우려는 경우에는 설계가 없으면 곤난하다.

마찬가지로 표의 수가 적고 표와 렬들사이의 관계가 그리 복잡하지 않은 자료기지는 특별한 설계가 없이도 만들수 있다. 또 이러한 자료기지는 상대적으로 변경하기도 쉽다. 그러나 업무용으로 리용하는 큰 자료기지들을 만들자면 반드시 면밀히 분석되고 타산된 설계가 있어야 한다.

일단 자료기지설계를 완성한 다음 그것을 변경하는것은 매우 어려운 일이다. 그것은 건물의 골조가 완성된 다음 구조를 변경하는것이 어려운것과 마찬가지이다. 자료기지의 구조를 변경하는것이 쉽다면 설계문제가 그다지 중요하지 않을것이다. 마구 만들어놓고 필요할 때 취미에 따라 고치면 그만이기때문이다. 그러나 완성된 구조에 변경을 가하는것은 새로 설계를 하는것보다 더 힘들수 있다. 자료기지에만 국한되는것이 아니라 일반적으로 복잡한 체계를 만드는데서 설계가 기본이다.

1) 요구사항분석

자료기지설계에서 가장 중요한 부분은 요구사항분석이다. 업무분석이라고도 하는 이 작업이 잘못되면 자료기지구조에 미흡한 부분이 생기고 미숙한 정찰자료에 의거한 군사작전과 마찬가지로 실패의 운명을 면치못한다.

자료기지의 목적과 용도의 결정

요구사항을 분석하기 위한 첫 단계는 자료기지의 목적과 용도를 결정하는것이다. 례를 들어 상업부분에서 판매관리자료기지를 구축한다고 할 때 그것을 필요로 하는 사람들과 기관들의 요구는 각이하다. 어떤 사람은 단순히 장부에 수자들을 기입하고 금액을 집계하는것이 힘들어서 자료기지를 구축할수 있다. 이러한 경우는 자료기지보다 Excel과 같은 표처리프로그람을 리용하는편이 나을수 있다. 그러나 자료가 축적되여 수행속도가 느려진다거나 보다 다양하고 상세한 정보를 요구한다면 표처리프로그람의 기능만으로는 한계에 부닥치게 될것이다. 어떤 사람은 재고파악을 위해서 자료기지구축을 요구할수

있고 또 경영일군들은 판매관련자료들을 분석하여 수요를 예측하고 판매 및 경영전략을 수립하려 할수도 있다.

이와 같이 같은 주제의 자료기지를 만드는 경우에도 단순히 손작업을 콤퓨터화하려는 목적으로부터 보다 높은 차원의 수요에 이르기까지 그 목적은 참으로 다양하다. 요구자의 목적에 따라 자료기지의 구조는 상대적으로 간단해지든가 혹은 아주 복잡한 구조를 가질수 있으며 필요한 자료의 범위와 깊이도 달라진다.

실례에서는 학교에서 성적관리의 손작업을 콤퓨터화하기 위한 간단한 자료기지의 설계를 목적으로 한다.

목적에 따르는 업무분석

자료기지의 목적을 설정한 다음에는 그에 따르는 업무분석을 한다. 실례에서 취급하는 성적관리업무에 대한 분석결과는 다음과 같다.

① 기본적인 기능은 학생들의 성적을 기록, 유지, 관리하는것이다.
② 성적은 학생이 어떤 과목을 수강한데 대한 총화이다.
③ 성적은 하나의 과목을 수강하기 시작하여 한 학기동안 배운 결과로 평가한다.
④ 학과목의 수강과 성적평가는 한 학기를 기준으로 하며 한 학기는 15주이고 1년에 두 학기를 진행한다.
⑤ 성적은 삭제되지 말고 계속 유지되며 언제든지 성적자료를 볼수 있어야 한다.
⑥ 매 과목에는 담당교원이 있다. 한명의 교원이 여러개의 과목을 담당할수 있으며 또 한개의 과목을 여러명의 교원들이 담당할수도 있다.
⑦ 한 과목의 성적은 매 과목에 할당되여있는 학점에 당담교원이 부여한 성적등급을 수자로 환산하여 곱한 값으로 계산된다.
⑧ 총점은 매 과목의 성적을 합하여 총 수강한 학점수로 나눈 값이다.
⑨ 교원이 학생으로 되거나 학생이 교원으로 되는 경우는 없다.
⑩ 모든 학생은 학과에 소속되여있고 매 학과는 특정한 학부 혹은 단과대학에 속한다.
⑪ 모든 교원은 학과에 소속되여있다.

이와 같이 간단한 자료기지를 설계할 때에는 요구사항분석목록이 그리 길지 않지만 여러가지 복잡한 업무들이 얽혀있는 큰 규모의 자료기지에서는 요구사항에 대한 분석이 수십 혹은 수백페지에 달할수 있다. 요구사항분석을 세밀히 할수록 설계에서 놓치는 부분이 없게 된다.

업무분석단계에서는 될수록 자료기지의 구조에 대해서는 신경쓰지 말고 필요한 자료를 생각하는데 중심을 둔다. 요구사항분석을 한 다음에는 그것을 기초로 자료스케치단계에 들어간다.

2) 자료스케치 및 표설계

자료기지의 용도와 목적을 결정하고 요구사항에 대한 분석이 끝나면 필요한 자료의 범위가 나오게 된다. 이러한 자료의 범위를 주제별로 나누는것이 이 단계에서 할 일이다. 업무분석에서는 단지 필요한 자료를 추출하는데 중심을 두었다면 이제부터는 자료기지를 구상하는데 중심을 둔다. 그러나 아직은 콤퓨터를 가지고 작업하는것이 아니고 종이와 연필을 가지고 그려보는 단계로서 자료스케치과정이라고 말할수 있다.

이 단계에서 나눈 분류에 따라 표들이 구성되고 이 표들이 정규화를 거치면 최량화된 자료기지의 구조로 된다. 이 단계를 거치지 않고 경험적인 방법으로 적당히 표를 구성하는 경우가 종종 있는데 초기에는 문제점이 생기지 않을수 있지만 마지막에 설계를 고치지 않으면 안되는 처지에 빠질수 있다.

필요한 자료를 큰 분류로 나누기

설계에서는 반드시 크게 생각하고 그 다음 세부적으로 분석하는 습관을 지녀야 한다. 앞에서 고찰한 요구사항목록을 따라가면서 필요한 자료의 범위를 큰 선에서 분류한다. 우선 1번 항목을 살펴본다. 학생의 성적을 기록, 유지, 관리하기 위해서는 어떠한 분류가 필요할것인가? 우선《학생》이라는 분류가 반드시 필요할것이다. 마찬가지로《성적》이라는것도 필요하다.

다음은 2번 항목을 살펴보자. 《성적》은 이미 정의해놓았으므로 《과목》이라는 분류가 하나 더 추가된다.

3번 항목에서는 성적이 처리(평가)되여야 할 기간을 명시하였고 4번은 성적이 처리되여야 할 기간인 학기에 대한 제한요구가 명시되여있다.

학기에 대한 제한요구라는 분류는 성적관리에는 필요없다. 학기라는 분류는 어떤가? 좀 아리숭하지만 학기는 기본적으로 날자에 근거한 자료이므로 후에 성적처리를 할때 날자에 조건을 두면 될수 있을것 같다. 실례로 2007년 4월 1일부터 2007년 8월 31일까지의 성적을 한학기로 두면 될것이다. 이렇게 하면 단순히《성적》이라는 분류에 학기의 시작날자와 끝날자의 렬을 구성하기만 하면 된다.

그러나 학기를 날자에 의한 구별로만 생각하지 않고 학기 그자체를 《2007년 1학기》,《2007년 2학기》등으로 표시되는 문자렬자료라고 생각할수도 있다. 매 방식에 다 우결함이 있을것이므로 어느 방식이 좋겠는가를 생각해보아야 한다.

가장 큰 차이는 학기를 날자자료로 처리하는 경우 《학기》분류를 별도로 둘 필요가

없다는 점이다. 학기의 시작과 끝날자는 매해 갈기때문에 이 날자들로부터 학기문자렬을 추출할수 있기때문이다. 례컨대 2007년 4월 1일부터 2007년 8월 31일까지 진행된 학기는 마땅히 《2007년 1학기》이므로 날자자료로부터 해당한 학기를 얻는것은 어렵지 않다.

성적
학기시작날자
학기끝날자
과목
성적

학기를 날자자료로 처리한 경우

성적
학기
과목
성적

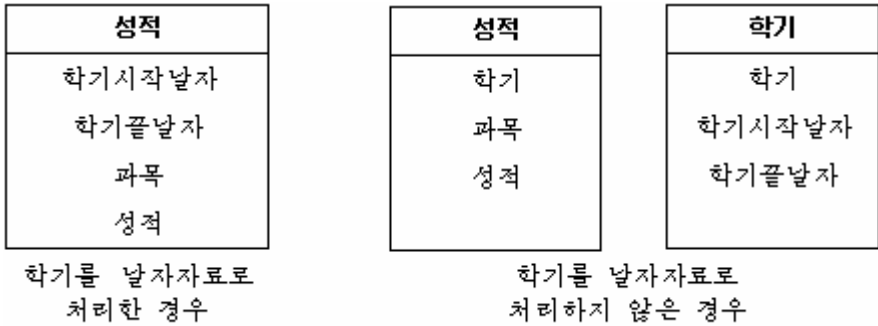
학기
학기
학기시작날자
학기끝날자

학기를 날자자료로 처리하지 않은 경우

그림 1-4. 학기처리의 두가지 방식

그렇지만 학기를 날자자료로 처리하지 않는 경우에는 해당한 학기의 시작과 끝날자가 포함된《학기》표를 만들어야 한다.(그림 1-4) 자료량이 조금 늘어나는 대신 학기별 성적처리를 할 때 학기를 구하기 위한 별도의 연산이 필요없고 단순히 학기표에서 해당한 학기를 선택하면 된다. 학기처리방식에 따르는 우점과 결함을 분석해보면 다음과 같다.

구분	학기를 날자자료로 처리할 경우	학기를 날자자료로 처리하지 않는 경우
우점	·별도의 《학기》표가 필요없다. ·학기를 등록하지 않아도 무한정 사용할수 있다.	·학기를 추출하는데 별도의 연산이 필요없고 《조건》만이 필요하다.
결함	·학기를 추출하는데 연산이 필요하며 다소 복잡한 식이 요구된다.	·별도의 《학기》표가 필요하다. ·학기를 별도로 등록해야 한다.

그림 1-5. 학기처리방식에 관한 우결함

어느것을 선택할것인가? 별도의 학기표가 필요없다는 우점보다 학기를 추출하는데 별도의 연산이 필요없다는것이 자료탐색속도의 견지에서 더 좋아 보인다. 그래서 학기를 날자자료로 처리하지 않기로 한다. 혹시 학기에 대한 날자가 필요할 때에는《학기》표를 참고하면 될것이다.

3번과 4번항목에서는 고려할 사항이 많았다. 세부적인 분석의 결과《학기》라는 자료분류가 필요하다는 결론을 내리게 되였다. 따라서《학생》,《성적》,《과목》,《학기》라는 큰 분류가 이루어졌다.

이제는 5번항목을 살펴보자. 보통 자료기지를 처음 배우는 사람에게 성적처리를 위한 성적표를 구성하라고 하면 다음과 같이 구성한다.

성적
과목
성적

그림 1-6. 실수하기 쉬운 성적표의 구성실례

이렇게 구성된 성적표는 시간(학기)의 구별이 없기때문에 한 학기가 지나면 모든 자료를 삭제하고 새로운 학기의 성적을 기록해야 한다. 이러한 점은 쉽게 빠질수 있는 함정이지만 5번항목의 도움으로 피할수 있다.

이번에는 6번항목을 살펴보자. 매 과목에는 담당교원이 있다고 하였으므로《교원》이라는 분류가 필요하다. 한명의 교원이 여러개의 과목을 담당할수 있고 한개의 과목을 여러명의 교원이 담당할수 있다고 하였다. 이러한 요소 역시 빠지기 쉬운 함정이다.

교원
교원번호
이름
학과
담당과목

교원표

과목
과목번호
과목명
교원번호

과목표

그림 1-7. 실수하기 쉬운 교원표의 구성실례

그림 1-7과 같이《교원》표를 구성하면 한명의 교원이 한가지 과목밖에 담당할수 없다. 만약 한명의 교원이 여러 과목을 담당하려면 다음과 같은 자료입력결과가 나온다.

표 1-19. 잘못 구성된 교원표의 자료입력실례

교원번호	이름	학과	담당과목
1	김동길	정보체계학과	정보체계개론
2	최영일	건축학과	건축시공학
⋮	⋮	⋮	⋮
2	최영일	건축학과	건축설비

앞의 그림에서 교원번호가 2번인 최영일이라는 레코드를 생각해보자. 담당과목을 《교원》표에 포함하고있기때문에 한 사람이 두개 과목을 담당할 경우 교원번호가 중복된다. 이것은 교원과 과목간의 관계를 잘못 파악했기때문에 파생되는 결과이다. 한명의 교원이 여러 과목을 담당할수 있고 한 과목을 여러 교원이 담당할수 있으므로 이것을 실현하기 위해서는 《교원》과 《과목》을 분리시켜야 한다.

다음으로 6, 7, 8번항목을 살펴보자. 지금 단계에서 6, 7, 8번항목만으로는 별도의 분류를 생각할수 없으므로 다음으로 넘어간다.

9번항목을 보자. 교원에 대한 정보로는 공민증번호, 학과, 이름, 주소, 전화번호 등이 될수 있다. 학생의 경우도 마찬가지이다. 교원과 학생은 그 렬구성이 같거나 비슷하지만 교원이 학생으로 되거나 학생이 교원으로 되는 경우가 없으므로 별도의 구분이 필요하다.

10번과 11번항목에 의해 《학과》라는 분류를 첨부한다.

지금까지 만들어진 분류는 《학생》, 《성적》, 《과목》,《학기》,《교원》,《학과》이다.

지금까지 자료를 분류별로 나누는 작업을 완성하였다. 이 분류에 따라 표들을 구성하게 된다.

표의 마당(렬)구성

지금까지 자료분류에 따라 구성한 표들에 대하여 필요한 마당(렬)들을 결정한다. 표에서 렬을 이루는 마당들을 구성하려면 크게 분류된 자료의 속성들을 생각해보면 된다. 례를 들어 《학생》의 경우 《한 학생을 알기 위해서는 어떤 정보가 필요한가?》라는 물음을 제기하고 그에 대한 답을 찾는것이 바로 학생에 대한 속성 즉 《학생》표의 마당을 구성하는것으로 된다. 한 학생을 알기 위해서는 이름, 공민증번호, 성별, 나이, 주소, 전화번호, 학과 등이 필요하다. 류사한 사고방식으로 나머지 표들에 대한 마당들을 구성해보자.

학생
공민증번호
이름
성별
나이
주소
전화번호
학과

교원
공민증번호
이름
성별
나이
주소
전화번호
학과

과목
과목명
담당교원
학점
학과명

학과
학과명
단과대학/학부명

성적
학기
학생
과목명
성적

학기
학기
학기시작
학기끝

그림 1-8. 자료분류별 마당의 구성상태

표손질: 기본열쇠설정과 외부열쇠의 리용

다음 단계는 앞 단계에서 작성된 표들에 기본열쇠를 할당하는 과정이다. 우선 학생표부터 살펴보자. 학생표의 경우 공민증번호가 기본열쇠로 될수 있다. 그렇지만 보안관계상 공민증번호를 기본열쇠로 사용하지 않는것이 관례이므로《학생번호》라는 마당을 새로 삽입하기로 한다.

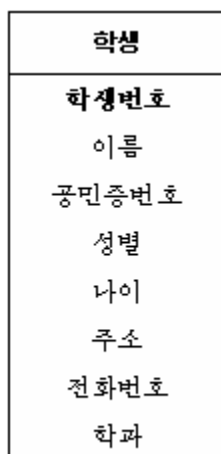



그림 1-9. 학생표에 기본열쇠를 할당한 결과

다음은 교원표인데 학생표에서와 같이 교원번호를 부여하여 기본열쇠로 한다.

그림 1-10. 교원표에 기본열쇠를 할당한 결과

다음 과목표를 살펴보자. 과목표에서는 어느 렬을 기본열쇠로 할것인가. 과목명을 기본열쇠로 할수 있을것 같지만 같은 과목을 여러명의 교원이 담당하는 경우가 있으므로 과목명과 담당교원을 조합해야 한다. 따라서 과목명과 담당교원의 조합에 의한 과목코드렬을 기본열쇠로 부가한다.

과목코드	과목명	담당교원	학점
1	정치경제학	리국범	2
2	정치경제학	최설송	2
3	생 물 학	박영호	1
4	정치경제학	김수련	2
	⋮	⋮	⋮

과 목 표

학생	과목코드(과목명/담당교원)	성적
박영철	1(정치경제학/리국범)	A
최성림	2(정치경제학/최설송)	B
김성호	3(생물학/박영호)	C
최일천	4(정치경제학/김수련)	D

성 적 표

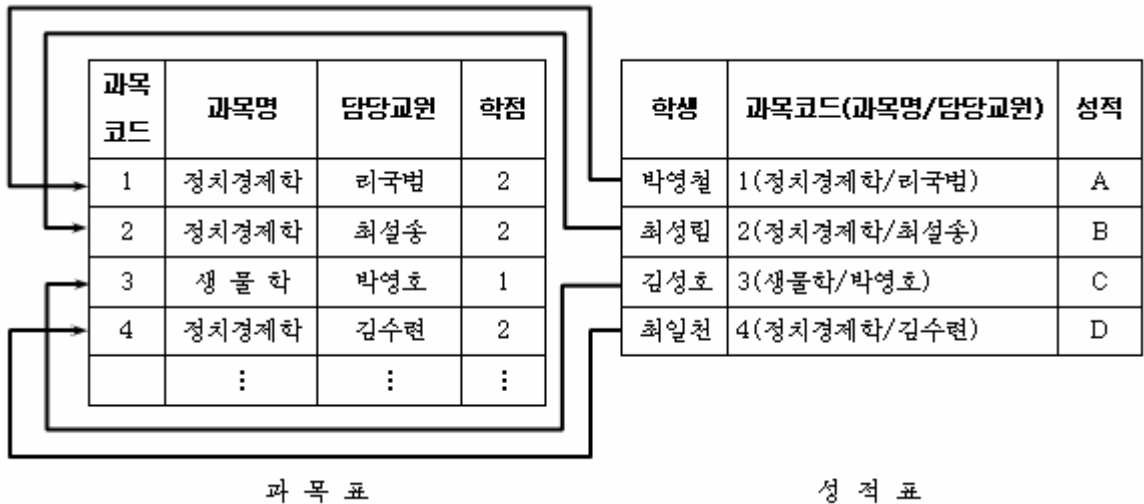



그림 1-11. 과목명과 담당교원의 조합에 따라 과목코드를 부여하는 경우

그림 1-11은 과목명과 담당교원의 조합에 따라 과목코드를 부여한 경우 자료입력을 예상한것이다. 과목표를 보면 정치경제학과목은 리국범, 최설송, 김수련이라는 3명의

교원이 담당하고있다. 성적표로부터 학생에 따라 같은 과목이라도 어느 교원한데서 배우고성적을 받았는가를 명확히 알수 있다.

과목표에서 담당교원은 교원표의 교원번호를 참고하므로 렬이름을 교원번호로 바꾼다. 그림 1-12에 완성된 과목표의 구성을 보여주었다.

그림 1-12. 과목표에 기본열쇠를 할당한 결과

다음으로 성적표에 대하여 보자. 성적표의 경우에는 기본열쇠로 될만한 렬이 없다. 그러나 학기, 학생번호, 과목명이 조합되면 기본열쇠가 될수 있다. 이것은 한 학기에 한 학생이 둘이상의 같은 과목을 수강하지 않는다는 사실에 기초한것이다. 여기서도 학생렬은 학생표의 기본열쇠인 학생번호로, 과목명은 과목표의 기본열쇠인 과목코드로 이름을 바꾼다.

그림 1-13. 성적표에 기본열쇠를 할당한 결과

이번에는 학과표를 살펴보자. 학과표에는 렬이 두개밖에 없다. 당연히 학과명이 기본열쇠로 될것이다.

그림 1-14. 학과표에 기본열쇠를 할당한 결과

다음은 학기표에 대하여 알아본다. 학기표에서 기본열쇠가 될수 있는 렬은 학기이다.

그림 1-15. 학기표에 기본열쇠를 할당한 결과

마지막 작업은 매 표에서 외부열쇠(참조열쇠)의 이름을 손질하는것이다. 례를 들어 그림 1-16과 같이 교원표의 외부열쇠이름은 《학과》이고 학과표의 기본열쇠이름은 《학과명》으로 설정되였다고 하자. 사람이라면 그 정도로 큰 무리가 없겠지만 콤퓨터는《학과》와 《학과명》을 전혀 다른것으로 인식하기때문에 두 표사이의 련결을 보장할수 없게 된다. 그러므로 외부열쇠의 렬이름을 참조받는 표의 기본열쇠이름과 정확히 일치시켜야 한다.

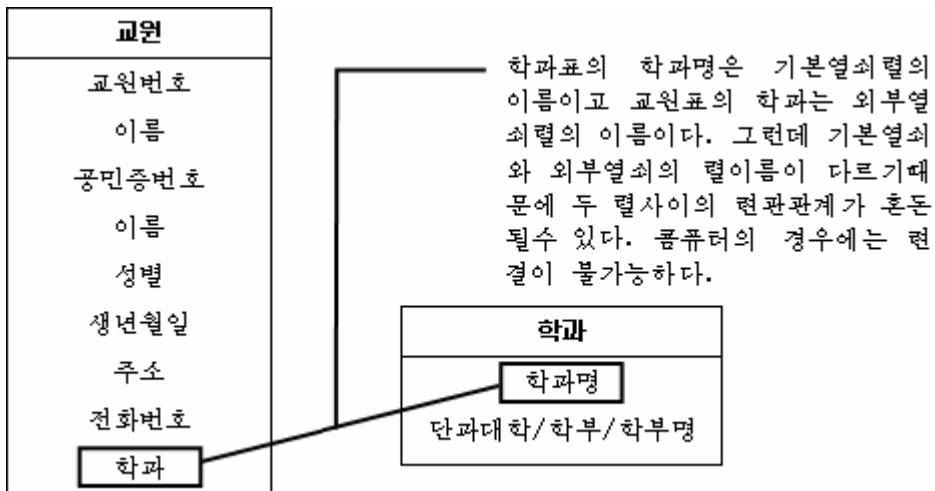



그림 1-16. 교원표과 학과표

학생표의 학과렬과 교원표의 학과렬이름을 다같이 《학과명》으로 일치시킨다.

지금까지 기본열쇠를 할당하고 표를 손질한 결과는 다음과 같다.

학생
학생번호
이름
공민증번호
이름
성별
생년월일
주소
전화번호
학과명

교원
교원번호
이름
공민증번호
이름
성별
생년월일
주소
전화번호
학과명

과목
과목코드
과목명
교원번호
학점
학과명

학과
학과명
단과대학/학부명

학기
학기
학기시작날자
학기끝날자

그림 1-17. 표에 기본열쇠를 할당한 결과

정규화

표에 기본열쇠까지 할당하였으므로 이제는 어느정도 자료기지의 모습을 갖추었다고 볼수 있다. 이제는 자료기지를 정규화하여 최량화하는 작업을 해야 한다.

▲ 1차정규화

1차정규화의 요구를 만족시키자면 표의 마당들이 모두 단일한 자료항목(혹은 가질수 있는 최소한의 자료)만을 포함하여야 한다. 학생표와 교원표에서 주소렬을 도, 시, 군/구역으로 갈라야 한다. 그러나 《도, 시, 군/구역별로 학생들의 성적을 종합하라.》와 같은 질문은 큰 의의가 없기때문에 편리상 주소렬을 도, 시, 군/구역으로 분리하지 않고 하나로 사용할수 있다.

▲ 2차정규화

제2정규형에서 매 마당들은 기본열쇠전체에 론리적으로 종속되여야 한다. 정규화를 위해서는 표들에 자료를 입력해보는것이 따져보기 쉽다. 우선 학과표부터 입력해본다.

학과표

학과명	단과대학/학부명
로보트공학과	기계기술대학
쏘프트웨어공학과	정보과학기술대학
인공지능학과	정보과학기술대학
정보체계학과	공업경영학부

학생표

학생번호	이름	공민증번호	성별	생년월일	주소	전화번호	학과명
1	박영철	2351341	남	19	평양시 중구역	01-352-2354	정보체계학과
2	최성림	0394852	녀	19	평양시 대성구역	01-521-0249	콤퓨터공학과
3	김성호	3567579	남	20	함남도 함흥시	02-721-2373	인공지능학과
4	최일천	2655757	남	25	평남도 안주시	05-623-6248	콤퓨터공학과

교원표

교원번호	이름	공민증번호	성별	생년월일	주소	전화번호	학과명
1	리국범	2353412	남	39	평양시 중구역	01-425-2354	로보트공학과
2	최설송	0395678	남	36	평양시 서성구역	01-346-0249	콤퓨터공학과
3	박영호	3523454	남	40	평양시 대성구역	01-432-2375	정보체계학과
4	김수련	2634567	녀	35	평양시 평천구역	01-756-5673	정보체계학과

지금까지의 표들은 모두 제2정규형을 만족한다. 다음은 과목표이다. 과목표에 자료를 입력한다.

과목표

과목코드	과목명	교원번호	학점
1	철학	5	2
2	철학	6	2
3	수학	9	2
4	수학	10	2

과목표의 경우 과목코드는 과목명과 담당교원(교원번호)의 조합이다. 그러므로 과목명은 기본열쇠에 완전함수종속이 아니다. 이러한 표에서 과목명과 같이 기본속성을 나타내는 렬에 중복이 일어났을 때에는 정규형규칙을 무시하고 사용하는 경우가 있는데 그러면 후에 통계자료를 뽑을 때 심각한 오유에 직면하게 된다.

례를 들어 과목코드 5번을 추가한다고 생각해보자. 과목명은 수학이고 교원번호는 11번이다. 그런데 사용자가 실수로 《소학》이라고 입력하는 경우 소학이라는 과목이 없음에도 불구하고 아무런 제한도 없이 등록된다. 후에 과목별(과목코드별이 아님) 총 수강인원수를 구할 때 《수학》과 《소학》은 엄연히 다른 과목으로 취급되므로 잘못된 통계자료를 얻게 된다.

따라서 과목표를 다음과 같이 분리한다.

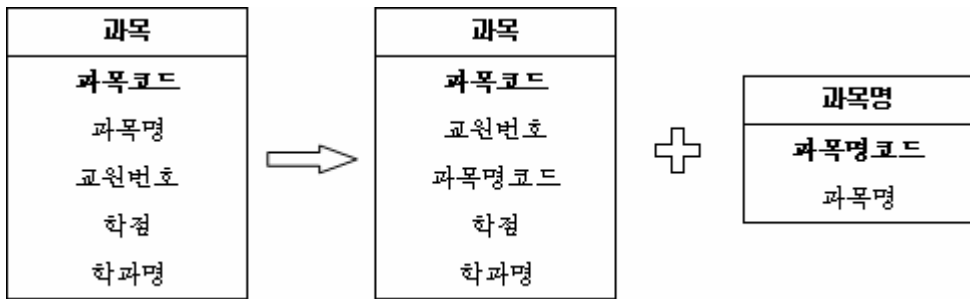



그림 1-18. 제2정규화리론에 의해 과목표를 과목표와 과목명표로 분리한 결과

이러한 구조에서 과목을 입력하려면 우선 과목명을 등록하여야 한다. 그리고 과목표에서 과목명코드를 입력할 때에는 참조의 완정성규칙에 의해 과목명표의 과목명코드값외의 값은 입력할수 없게 된다.(참조의 완정성규칙이란 자료의 정확성 혹은 완전무결성에 관한 규칙으로서 다음 소절에서 학습한다.) 따라서 앞의 실례와 같은 오유를 사전에 방지할수 있게 된다. 이처럼 정규화를 하면 자료기지의 크기가 작아지고 최량화될뿐만아니라 자료의 완정성도 한층 강화된다.

❖ 정규화의 목적 ❖

오늘날 하드웨어의 가격이 눅어지고 성능은 급격히 높아지고있다. 지금의 탁상형콤퓨터는 몇년전 봉사기의 자료처리량과 맞먹거나 그 이상이다.
정규화의 기본목적의 하나는 중복된 자료를 제거하여 저장공간(디스크 및 메모리)을 효률적으로 사용하고 자료관리성능을 높이자는데 있다. 그렇지만 하드웨어가격이 눅어지고 성능이 높아진 지금의 환경에서는 이러한 저장공간의 효률적인 사용을 위한 정규화보다 자료입력의 완정성을 위한 정규화의 의미가 더 강하다.

다음은 학기표이다. 학기표에 자료를 입력해보자.

학기	학기시작날자	학기끝날자
2002년 1학기	2002-04-01	2002-09-20
2002년 2학기	2002-10-01	2002-03-20
2003년 1학기	2003-04-01	2003-09-20
2003년 2학기	2003-10-01	2003-03-20
2007년 1학기	2007-04-01	2007-09-20
2007년 2학기	2007-10-01	2007-03-20

학기표에는 전체열쇠와 무관계한 마당이 없으며 제2정규형을 만족한다.

마지막으로 성적표를 살펴보자. 성적표에서 기본열쇠는 학기, 학생번호, 과목코드이므로 성적렬에서 중복이 있는가를 따져보면 된다.

성적렬을 보면 일부 값들이 중복되여있는것을 볼수 있다. 이것도 중복된 값이므로 정규화의 대상에 들지만 렬의 크기가 1이므로 단순히 저장용량을 최량화하기 위해 정규화를 한다는것은 적절한 의미가 아니다.

학기	학생번호	과목코드	성적
2007년 1학기	1	1	5.0
2007년 1학기	1	2	4.4
2007년 1학기	1	3	4.8
2007년 1학기	1	4	4.9
2007년 1학기	1	5	4.1
2007년 1학기	2	1	2.8
2007년 1학기	2	2	3.9
2007년 1학기	2	3	3.1
2007년 1학기	2	4	4.6
2007년 1학기	2	5	5.0
2007년 1학기	3	1	3.7
2007년 1학기	3	2	4.5
2007년 1학기	3	3	3.9
2007년 1학기	3	4	3.5
2007년 1학기	3	5	4.3
2007년 1학기	4	1	3.4
2007년 1학기	4	2	4.6
2007년 1학기	4	3	4.2
2007년 1학기	4	4	2.5
2007년 1학기	4	5	3.4
2007년 2학기	1	1	4.4
2007년 2학기	1	2	3.7
2007년 2학기	1	3	4.9
2007년 2학기	1	4	4.9
2007년 2학기	1	5	3.8

그렇다면 무엇을 위해서 정규화를 할것인가? 앞에서 요구사항분석목록의 7번항목과 8번항목을 살펴보자.

⋮
⑦ 한과목의 성적은 매 과목에 할당되여있는 학점에 담당교원이 부여한 성적등급을 수자로 환산한것을 곱하여 계산한다.
⑧ 총점은 매 과목의 성적을 합하여 총 수강한 학점수로 나눈 값이다.
⋮

7번과 8번항목은 성적계산규정을 설명하고있다. 7번항목을 보면 《학점에 담당교원이 부여한 성적등급을 수자로 환산한것을 곱한다.》는 규칙으로 한과목의 성적을 계산하는 방식을 설명하고있고 8번항목의 경우 총점을 계산하는 규칙에 대하여 설명하고있다.

일반적으로 성적평가는 어떤 최고점수를 기준으로 과제수행정형 10%, 실험 및 학과설계(학과론문) 10%, 학기말성적 70%, 출석률 10% 등 세부평가항목을 세운 다음 매 항목에 따라 점수를 매기고 더하는 방법으로 진행한다. 여기서는 100점 만점으로 성적을 평가하고 일정한 점수의 덩어리에 A, B, C, D, E의 다섯 준위로 성적등급을 부여하는 경우를 고찰한다. 그리고 성적을 계산할 때에는 다시 A는 4.5, B는 3.3과 같은 형식으로 수자로 변환한다.

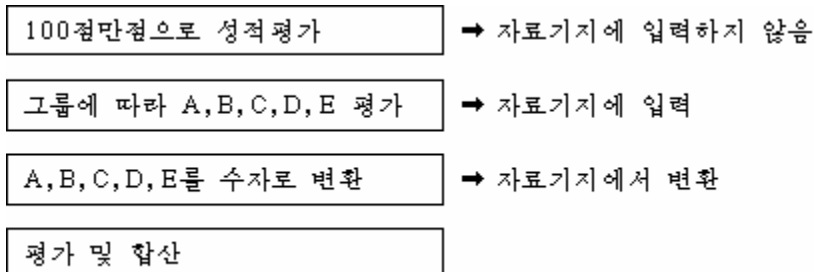



그림 1-19. 성적평가단계

앞의 그림은 일반적인 성적평가단계를 나타낸것이다. 그런데 이러한 점수를 입력하는 과정을 살펴보면 100점 만점으로 성적평가를 하고 그룹을 나눌 때까지의 작업은 사람이 하는것이 보통이다. 왜냐하면 성적은 A, B, C, D, E의 다섯등급으로 나눌수 있지만 매 성적등급에 대한 비률은 담당교원의 의도와 결심에 따라 다르기때문이다. 학생들이 배워준 내용에 대한 인식정도가 좋다면 A, B의 비률이 높을수 있고 반대의 경우는 그 비률이 낮을수 있다. 이러한 기준은 담당교원마다 다를수 있으므로 이것을 자료기지화하는것은 좋은것이 못된다.

그러나 A, B, C, D, E라는 성적등급은 일정한 수자와 1:1로 대응된다. 교원이 마

음대로 A에 4.3, B에 4.2를 부여할수 없다. 그러한것은 그룹을 만들 때 결정된다. 따라서 A, B, C, D, E와 그에 대응되는 수자(점수)의 관계는 자료기지에서 처리할수 있다. 이러한 리유로부터 성적표를 정규화대상에 포함시켜야 한다. 성적표는 성적표와 성적등급표로 분리할수 있다.

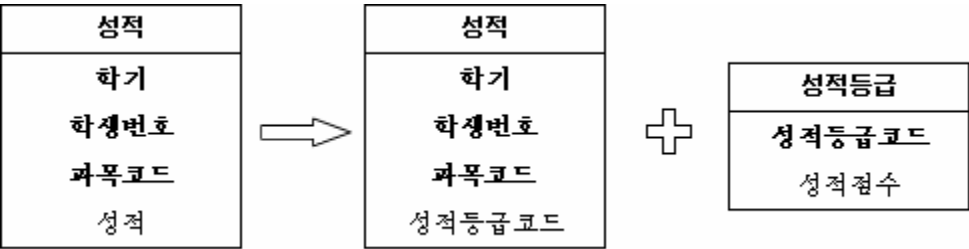



그림 1-20. 성적표를 제2정규화리론에 의해 성적표과 성적등급표로 분리한 결과

표를 이와 같이 구성해놓으면 후에 통계자료를 구할 때 성적등급에 따라 성적표에서 성적점수를 추출하여 쉽게 계산할수 있다.

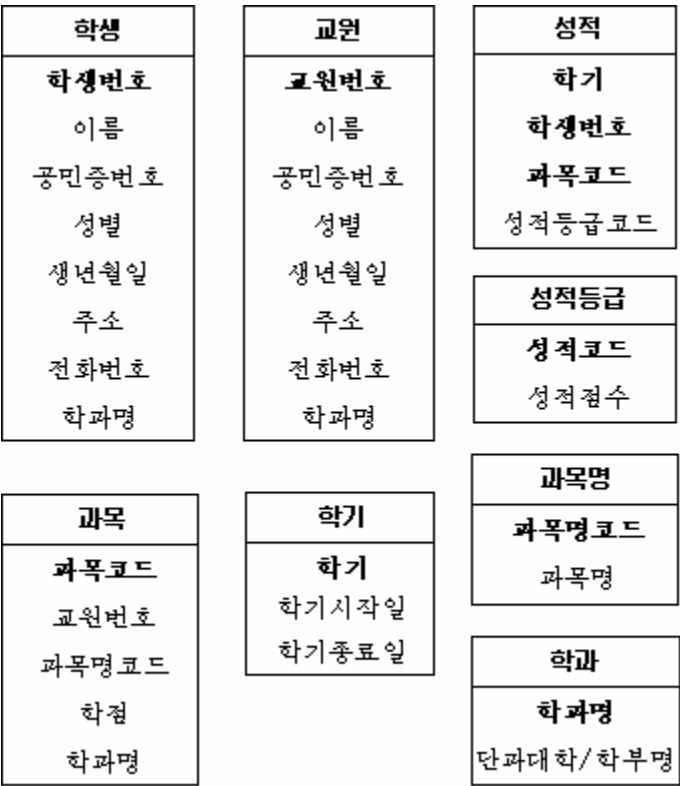



그림 1-21. 제2정규형을 만족하는 표구조

▲ 3차정규화

이제는 제3정규화를 통하여 표의 구조를 최량화하는 단계이다. 제3정규화는 기본열쇠와 관련이 없는 마당 즉 기본열쇠에 이행적함수종속인 마당은 포함하지 말것을 요구한다. 이것은 3정규형에서는 한 마당의 값을 알았을 때 다른 마당의 값을 계산 혹은 추측할수 없도록 마당들이 서로 독립적이여야 한다는것을 의미한다. 3정규화에서는 기본열쇠와 무관계한 마당을 없애거나 별도의 표로 분리한다. 앞에서 본 2정규형을 만족하는 표들은 3정규화의 요구도 만족한다.

❖ 주의: 의도된 비정규화 ❖

3정규화이상 지어는 3정규화나 2정규화까지도 필요에 따라서는 하지 않을수 있다. 2정규형의 포기는 극히 드물지만 3정규형을 포기하는 경우는 상당히 많다. 그 리유는 정규화로 하여 오히려 자료기지의 성능을 떨어뜨리는 경우가 있기때문이다. 우리가 명심할것은 《최적의 정규형이 최상의 자료기지모형이 아니다.》라는것이다.
그러나 일반적으로 옳바른 자료기지정규화는 자료기지의 성능을 향상시키고 자료를 보다 더 효률적으로 관리할수 있게 한다. 그러므로 비정규화를 해야 할 마당이 있는 경우에도 우선 정규화를 수행해보고 정규화와 비정규화의 우단점을 따져보고 정규화를 할것인가 비정규화를 할것인가를 결정해야 한다.

1.4.2. 참조의 완정성

많은 기관, 기업소들에서는 자료기지의 정보를 기초로 앞으로의 계획도 세우고 방향도 설정한다. 이러한 자료기지의 정보를 믿을수 없다면 차라리 사용하지 않는것만 못할수 있다. 때문에 자료기지내부의 정보에 대한 믿음성문제는 그 중요성을 구태여 강조할 필요가 없을것이다. 자료기지의 **완정성**이란 자료기지에 들어있는 자료의 믿음성을 보장하는 문제와 관련된 지표이다. 어떤 사용자가 고의적으로 잘못된 정보를 입력하는것까지 방지할수는 없어도 최소한 사용자의 실수로 인한 오유는 방지할수 있어야 한다.

자료기지의 완정성을 담보하기 위해서는 자료기지설계에서 자료의 유효성과 정확성을 검사할수 있도록 만든 **규칙**(rule)을 정확히 지켜야 한다. 완정성규칙에는 일반완정성규칙과 자료기지특정의 완정성규칙이 있다.

1) 일반완정성규칙

관계모형은 모든 자료기지들에 적용하는 두가지 일반완정성규칙을 가진다.

- 실체(개체)완정성규칙(Entity integrity rule)
- 참조완정성규칙(Referential integrity rule)

실체완정성규칙은 기본열쇠에 NULL이 포함될수 없다는것을 규정하고있다. NULL이 행을 유일하게 참조하는데 리용될수 없다는것은 누구에게나 명백할것이다. 복합열쇠의 경우에도 NULL이 포함되지 말아야 한다.

참조완정성규칙은 자료기지가 임의의 정합되지 않는 외부열쇠를 포함하지 말아야 한다는것이다. 다시말하여 외부열쇠에 의한 모든 참조는 참조되는 표에 실제적으로 존재하는 행들을 식별하는 기본열쇠들을 지적하여야 한다.

참조완정성규칙은 또한 참조할 기본열쇠가 없는 외부열쇠를 남겨둠으로써 그 외부열쇠에 의해 참조되는 행에 대한 변경 혹은 삭제를 막기 위하여 정확한 조치가 취해져야 한다는것을 의미한다. 이것은 다음의 방법들로 처리될수 있다.

- 이러한 변화들이 허락되지 않을수 있다.
- 참조되는 기본열쇠를 포함하는 행의 삭제가 의존하는 표들에서 련결되는 모든 행들을 삭제하도록 종속련결할수 있다.
- 의존하는 외부열쇠값들을 NULL로 설정한다.

구체적인 조치는 그때의 정황에 따라 달라진다. 많은 관계형자료기지체계들은 참조완정성규칙에 위반되는 시도들을 처리하기 위한 방법들을 지원해준다. 례를 들어 적당한 표에서 찾을수 없는 외부열쇠를 포함한 행을 삽입하려는 시도는 다음과 같은 SQL의 EXCEPTION통보문으로 끝난다.

```
INSERT statement conflicted with COLUMN FOREIGN_KEY constraint
'FK_CONTACTS_ADDRESS_INFO'. The conflict occurred in database 'CUSTOMERS',
table 'INVENTORY', column 'id'.
```

2) 자료기지특정의 완정성규칙

자료기지특정의 완정성규칙은 특정한 자료기지의 목적과 용도에 따르는 기타 완정성규칙을 말한다. 그것들은 보통 응용분야의 업무론리에 따라 처리된다.

실례로 우리가 앞에서 본 학생표나 교원표에서 성별마당에 내용상 남성과 녀성이 아닌 어떤 다른 값(례컨대 가상적으로 누군가가 심심풀이로 중성)을 입력하면 어떻게 될것인가. 그것은 단지 한 사람의 성별만 잘못되는것이 아니라 성별에 따르는 여러가지 통계를 낼 때 잘못된 결과를 내게 된다. 이와 같이 업무내용상 모순되는 자료의 입력을 막기 위한 규칙들이 특정한 자료기지에 따르는 **특수규칙**이며 **업무규칙**이라고도 한다.

자료기지특정의 완정성규칙에 관한 몇가지 실례는 다음과 같다.

- 성별마당에는 《남자》 혹은 《녀자》의 값만이 들어갈수 있다.
- 사람의 나이는 200살을 넘을수 없다.
- 수량에는 부수가 올수 없다.
- 같은 상점에서 하루에 한가지 상품을 한번이상 구입하지 않는다. 혹시 오전에 한번, 오후에 한번 구입했다고 해도 그 구입량을 하나로 합쳐 한번에 구입한것으로 한다.

이러한 실례들은 매우 상식적인것이며 이 규칙들을 지키지 않으면 자료기지는 신뢰성을 담보할수 없다. 이러한 업무규칙들을 정확히 적용하기 위해서는 자료기지와 관련된 업무내용과 규정세칙들을 정확히 알고있어야 한다.

완정성제약의 대부분이 SQL트리거들에 의해 처리될수 있지만 일부는 자바업무론리에 의해 처리된다. 트리거란 자료기지에 대한 삽입 혹은 변경과 같은 사건들에 의해 시동되는 SQL수속들을 말한다.

1장에 대한 요약

이 장에서는 관계형자료기지관리체계가 어떻게 일하는가에 대해서와 자료기지설계를 위한 정규형의 상식적인 응용에 대하여 실례를 들어 고찰하였다.

기본적인 론제들은 다음과 같다.

- 자료기지와 그를 구성하는 표의 생성과 정규화
- 표들을 련결하기 위한 기본열쇠와 외부열쇠의 리용
- 구조화질문언어
- 거래와 거래관리에 대한 리해
- 정규화규칙의 적용
- 참조완정성규칙

2장에서는 관계형자료기지에서 작업할 때 리용하는 SQL언어를 취급한다.

제 2 장. SQL 의 기초

1장에서도 언급한 바와 같이 명백히 정의된 자료조작언어는 그 어느 자료기지체계에서나 중요한 지위를 차지한다. 코드는 자료조작과 정의, 뷰정의, 완정성제약조건, 거래한계와 인증에 관한 포괄적인 지원을 포함하는 언어에 대한 요구를 제기하였다. 또한 언어는 관계모임에서처럼 자료를 삽입, 갱신, 검색, 삭제할수 있는 능력이 있어야 한다고 지적하였다.

현재 모든 자료기지들에 적용되여있는 그러한 언어가 구조화질문언어이다. 이 장에서는 SQL에 대하여 포괄적으로 개괄한다.

SQL은 자료기지질문에 리용될뿐만아니라 자료기지체계의 전체기능을 조종하는데 리용된다. 이 각이한 기능들을 지원하기 위하여 다음과 같은 부분언어들이 부속되여있다.

- 자료정의언어(DDL)
- 자료조작언어(DML)
- 자료질문언어(DQL)
- 자료조종언어(DCL)

자바나 대부분의 다른 콤퓨터언어들과 달리 SQL은 수속적인것이 아니라 서술적이다. 다시말하여 SQL에서는 어떤 과제를 수행하기 위하여 클라스를 작성하는것이 아니라 표를 갱신하거나 레코드묶음을 돌려주는 문을 내보낸다.

제1절. SQL자료형

SQL은 표 2-1에 보여준것처럼 여러가지 자료형들을 지원한다. SQL자료형에 대응하는 JDBC자료형들을 함께 주었다. 많은 SQL파생언어들이 각이한 방법으로 이 자료형들을 지원하기때문에 최대문자렬길이나 수값들 그리고 대형객체저장에 어느 자료형을 리용해야 하는가를 고려하면서 책을 읽는것이 중요하다.

표 2-1. SQL자료형과 자바에서의 대응하는 자료형

SQL자료형	Java자료형	설명
BINARY	byte[]	바이트형배렬. 2진대형객체들에 리용된다.
BIT	boolean	론리값형.(0 또는 1)
CHAR	String	고정길이문자렬. 길이가 n인 CHAR형에 대하여 자료기지관리체계는 사용하지 않는 기억기공간에 n개문자분의 공간을 고정적으로 할당한다.
DATETIME	java.sql.Date	"yyyy-mm-dd hh:mm:ss"형식의 날자와 시간.
DECIMAL	java.math. BigDecimal	임의의 정확도를 가진 부호있는 10진수. 이것은 BigDecimal이나 String을 리용하여 검색될수 있다.
FLOAT	double	류동소수점수. double형에 대응한다.
INTEGER	int	32비트 옹근수값
LONGVARBINARY	byte[]	가변길이문자렬. JDBC는 LONGVARBINARY를 입력흐름으로 검색할수 있게 한다.
LONGVARCHAR	String	가변길이문자렬. JDBC는 LONGVARCHAR를 입력흐름으로 검색할수 있게 한다.
NCHAR	String	민족문자유니코드 고정길이문자렬
NUMERIC	Java.math. BigDecimal	임의의 정확도를 가진 부호붙은 10진수. BigDecimal이나 String을 리용하여 검색될수 있다.
NTEXT	String	큰 문자렬변수. 큰 문자객체들에 리용된다.
NVARCHAR	String	민족문자유니코드 가변길이문자렬
REAL	float	단순정확도 류동소수점수
SMALUNT	short	16bit 옹근수값
TIME	java.sql.Time	java.util.Date를 둘러싸는 포장
TIMESTAMP	java.sql.Timestamp	java.util.Date와 개개의 나노초값의 합성
VARBINARY	Byte[]	바이트배렬
VARCHAR	String	가변길이문자렬. 길이가 n인 VARCHAR에 대하여 자료기지관리체계는 요구에 따라 n개 문자까지 할당한다.

대부분의 SQL파생어들에서는 MONEY나 CURRENCY와 같은 추가적인 자료형도 지원한다. 이런 자료형들은 자바에서 적절한 취득자(getter)메쏘드나 설정자(setter)메쏘드를 리용하여 처리한다.

임의의 SQL자료형의 자료는 getObject()메쏘드를 리용하여 검색할수 있다. 이러한 메쏘드는 특히 자료형을 알수 없는 경우에 실제적으로 쓸모가 있으며 응용프로그람의 어디서나 자료를 이끌어낼수 있다. 더우기 많은 자료형들은 JDBC가 필요한 자료형변환을 시도하므로 getString()과 기타 다양한 취득자메쏘드들을 리용하여 추출할수 있다.

제2절. 자료정의언어

SQL의 자료정의언어는 자료기지를 창조하고 변경하는데 리용된다. 다시말하여 DDL은 자료기지의 구조변경과 관계된다.

DDL의 기본지령들은 CREATE, ALTER, DROP이다. 이 지령들을 적용할수 있는 자료기지요소들과 함께 표 2-2에 보여주었다.

표 2-2. DDL지령

지령	자료기지	표	뷰	색인	함수	저장된 수속	트리거
CREATE	예	예	예	예	예	예	예
ALTER	아니	예	예	아니	아니	아니	아니
DROP	예	예	예	예	예	예	예

2.2.1. 자료기지와 표의 작성, 제거 및 변경

자료기지의 창조

자료기지를 창조하는데 리용하는 SQL지령은 다음과 같다.

```
CREATE DATABASE 자료기지이름;
```

대부분의 관계형자료기지관리체계는 기록파일이름과 같은 파라메터들은 물론 리용하는 파일이나 파일그룹들을 지적할수 있도록 지령의 확장판본들을 지원해준다. 만일 기초지령외에 더 많은 지령들을 리용하려면 해당한 관계형자료기지관리체계참고서들을 참조하기 바란다.

자료기지를 제거하는 DROP지령 역시 단순하다.

```
DROP DATABASE 자료기지이름;
```

표 작성

관계형자료기지에서는 자료를 표에 저장한다. 대부분의 자료기지는 서로 다른 표들을 수많이 포함할수 있으며 매 표는 프로그람에 따라 각이한 형의 자료들을 포함한다.

표를 작성할 때 매 렬의 자료형과 마당길이가 설정된다.

```
CREATE TABLE 표이름
   (렬이름 자료형[(크기)] [제약] [기정값], ...);
```

표를 작성할 때 매 렬의 자료형과 길이를 설정하는것과 함께 그 렬에 저장하는 자료에 적용되는 다양한 제약조건들을 지정한다. 이 제약조건들은 자료의 일관성과 정확성을 담보하기 위한 제한으로서 완정성제약이라고 부른다. 그것들은 다음과 같다.

- NULL 혹은 NOT NULL(Null값 허용상태)
- UNIQUE(유일성)
- PRIMARY(기본열쇠)
- FOREIGN(외부열쇠)

목록 2-1은 판매주문관리체계에서 주문자표를 작성하는 CREATE TABLE문을 보여주고있다.

표이름을 먼저 쓰고 괄호안에 렬이름들을 차례로 정의한다. 렬이름뒤에는 자료형과 제약조건들을 쓰는데 이 제약조건들은 생략할수도 있다. 매 렬에 대한 정의는 목록에서 보는바와 같이 반점으로 구분한다.

목록 2-1. CREATE TABLE문

```
CREATE TABLE CUSTOMERS
  (CUSTOMERS_ID    INTEGER       NOT NULL      PRIMARY KEY,
   FAMILY_NAME     VARCHAR(10)   NOT NULL,
   PERSONAL_NAME   VARCHAR(10)   NOT NULL,
   SEX             VARCHAR(2)    NOT NULL,
   STATE           VARCHAR(10)   NOT NULL,
   CITY            VARCHAR(10)   NULL,
   DISTRICT        VARCHAR(10)   NULL,
   DONG            VARCHAR(10)   NOT NULL,
   NEIGHBOR        VARCHAR(10)   NOT NULL);
```

목록 2-2의 실례는 외부열쇠생성에 대하여 보여주고있다. 외부열쇠로 정의된 렬 SIGNIFICANT_OTHER는 주문자표에 있는 매 항목들을 련결하는데 리용된다.

목록 2-2. 외부열쇠를 포함하는 표작성

```
CREATE TABLE SIGNIFICANT_OTHERS
  (CUSTOMER_ID          INT   NOT NULL    PRIMARY KEY,
   SIGNIFICANT_OTHER   INT,
   FOREIGN KEY(SIGNIFICANT_OTHER) REFERENCES CUSTOMERS);
```

표변경

ALTER TABLE지령은 주로 렬들을 추가, 변경 혹은 삭제하는데 리용된다.

례를 들어 주문자료에 전화번호를 보관하는 PHONE렬을 추가하기 위해서는 다음과 같은 지령을 리용한다.

```
ALTER TABLE CUSTOMERS ADD PHONE VARCHAR(20);
```

렬길이는 다음의 지령을 리용하여 변경한다.

```
ALTER TABLE CUSTOMERS ALTER PHONE VARCHAR(30);
```

끝으로 렬을 완전히 삭제하기 위한 지령은 다음과 같다.

```
ALTER TABLE CUSTOMERS DROP COLUMN PHONE;
```

표삭제

DROP지령을 리용하여 자료기지에서 표를 완전히 삭제할수 있다. 주문자료(CUSTOMERS)를 삭제하기 위한 지령은 다음과 같다.

```
DROP TABLE CUSTOMERS;
```

2.2.2. 뷰의 작성, 변경 및 제거

뷰는 표와 대단히 류사하다. 뷰도 표와 같이 다른 질문들에서 접근할 때 리용할수 있는 이름을 가진다. 실제상 뷰는 **림시표**(temporary table)라고 할수 있다.

뷰의 생성

뷰는 기초자료기지의 기본부분으로 작성되는것이 아니라 아래에 보여준것처럼 질문을 리용하여 만들어진다.

```
CREATE VIEW viewKim AS
SELECT *
FROM Customers
WHERE Family_Name = '김'
```

이제는 이 뷰를 보통의 표처럼 여기면서 질문을 실행할수 있다.

```
SELECT *
FROM viewKim
WHERE State = '평양시'
```

이 질문은 다음과 같은 결과모임을 돌려준다.

ID	Family_ Name	Personal_ Name	Sex	State	City	District	Dong	Neighbor
100	김	성철	남	평양시	평양	중구역	교구동	32
101	김	은희	녀	평양시	평양	보통강구역	대보동	17

뷰는 실제상 이름을 가진 결과모임에 불과하므로 여러개의 표들을 련결하여 뷰를 생성할수 있다. 여러개의 표로부터 자료를 추출하는 한가지 방도는 INNER JOIN을 리용하는것이다. 다음의 코드는 《Orders_by_Name》이라는 뷰를 생성하기 위하여 INNER JOIN을 어떻게 리용하는가를 보여준다.

```
CREATE VIEW Orders_by_Name AS
SELECT c.Family_Name + c.Personal_Name AS Name,
   COUNT(i.Item_Number) AS Items,
   SUM(oi.Qty * i.Cost) AS Total
FROM Orders o INNER JOIN Ordered_Items oi
```

```
    ON o.Order_Number = oi.Order_Number
    INNER JOIN Inventory i
    ON oi.Item_Number = i.Item_Number
    INNER JOIN Customers c
    ON o.Customer_ID = c.Customer_ID
GROUP BY c.Family_Name + c.Personal_Name
```

사실 SELECT문이 돌려주는 임의의 결과모임이 뷰를 생성하는데 리용될수 있다. 그것은 뷰생성에 단순한 SELECT문은 물론 중첩된 질문들, JOINS 혹은 UNIONS를 리용할수 있다는것을 의미한다.

뷰의 변경

뷰는 SELECT지령을 리용하여 작성되기때문에 새로운 SELECT지령을 내는 ALTER지령에 의해 변경될수 있다.

레를 들어 앞에서 작성한 뷰 viewKim을 변경하기 위해서는 다음과 같은 지령을 리용할수 있다.

```
ALTER VIEW viewKim AS
SELECT Family_Name, Personal_Name
FROM Customers
WHERE Family_Name = '김'
```

자료검색만이 아니라 행들을 갱신하거나 삭제하는데도 뷰를 리용할수 있다. 뷰는 진짜 표가 아니고 기본표로부터 유도된 가상표이므로 단순히 표를 보기 위한 한가지 수단이다. 그러나 뷰에서 갱신 및 삭제된 행들은 본래의 표에서도 갱신되거나 삭제된다. 다음의 실례는 앞에서 생성된 뷰에서 《김은희》의 인민반주소를 변경시키는것이다.

```
UPDATE viewKim
SET Neighbor = '25'
WHERE Family_Name = '김' AND Personal_Name = '은희'
```

주의: 뷰는 어떤 의미에서 이름을 리용하여 보관할수 있는 질문이다. 왜냐하면 자료기지관리체계는 일반적으로 뷰이름과 함께 뷰생성에 리용된 SELECT문을 련관시켜 뷰를 보관하며 사용자가 그 뷰에 접근하려고 할 때 그 SELECT를 실행한다. 결함은 뷰를 리용할 때마다 부가적인 처리시간이 요구된다는것이다.

제3절. 자료조작언어

자료조작언어(DML)는 표에 자료를 삽입하고 필요에 따라 자료를 수정하거나 삭제하는데 리용된다. 자료기지에서 자료를 다루는데 리용하는 SQL문은 다음과 같다.

- INSERT
- UPDATE
- DELETE

2.3.1. INSERT문

INSERT문은 표에 행을 삽입하는데 리용된다. 가장 단순한 형식으로 쓰면 한번에 한개 행을 삽입한다. INSERT문은 또한 SELECT문과 결합하여 다른 표들로부터 선택한 여러개의 행들을 삽입하는데도 리용할수 있다. 이 지령은 표에 전체 행을 삽입하는데만 리용할수 있고 개별적인 마당들을 삽입하는데는 리용할수 없다.

INSERT문의 기본형식은 다음과 같다.

```
INSERT INTO 표이름 (렬이름1, 렬이름2, ...) VALUES (값1, 값2, ...);
```

주문자표에 이름과 주소정보를 삽입하기 위한 INSERT문의 리용실례는 다음과 같다.

```
INSERT INTO CUSTOMERS
(Family_Name, Personal_Name, Sex, State, City, District, Dong, Neighbor)
VALUES
('최', '영미', '녀', '함경북도', '청진시', '수남구역', '말음1동', '10');
```

렬이름들은 렬정의순서대로 지적하여야 한다.

표의 렬순서를 아는 경우에는 다음과 같은 속기형식을 리용할수도 있다.

```
INSERT INTO CUSTOMERS VALUES
('최', '영미', '녀', '함경북도', '청진시', '수남구역', '말음1동', '10');
```

우에서 주문자표를 정의할 때 City마당과 District마당은 NULL값을 허용하도록 정의되였다. NULL값을 삽입하는 정확한 방법은 다음과 같다.

```
INSERT INTO CUSTOMERS VALUES
('리', '철민', '남', '평안북도', '신의주시', NULL, '역전동', '45');
```

주의: 문자렬자료는 실레에서 보여준것처럼 단일인용부호(')안에 써야 한다. 수값자료는 인용부호가 없이 규정된다.

INSERT문으로 표에 자료를 삽입할 때 사용자들이 지켜야 할 몇가지 규칙들은 다음과 같다.

- 사용자가 리용하는 렬이름들은 렬에 정의된 이름과 일치해야 한다.
- 삽입하는 값들은 삽입하려는 행에 정의된 자료형과 일치해야 한다.
- 자료의 크기는 렬크기를 초과하지 말아야 한다.
- 렬에 삽입하는 자료는 렬에 할당된 자료제약조건에 맞아야 한다.

이 규칙들을 지키지 않으면 SQL은 례외처리를 일으키게 된다.

INSERT … SELECT의 리용

INSERT문은 한 표에서 다른 표로 일부 자료들을 복사하는데도 널리 리용할수 있다. 이 경우에 INSERT문은 요구하는 레코드들을 선택하기 위하여 SELECT문과 결합하여 리용된다. 이 방법을 리용하면 전체 처리가 자료기지관리체계내부에서 진행되므로 레코드를 검색하고 그것을 외부에서 다시 삽입하는 부가처리를 피할수 있는 우점이 있다.

실례로 주문자표에서 주문자의 성과 이름만을 추출하여 새로운 표를 만들고 싶은 경우에 INSERT ... SELECT를 리용할수 있다. 본래의 주문자표에서 성과 이름을 얻어내는데 SELECT문을 쓰고 얻은 결과를 새로운 표에 삽입하는데 INSERT문을 리용한다.

```
INSERT INTO Names
SELECT Family_Name, Personal_Name FROM Customers;
```

이 지령은 자료기지관리체계에 내부적으로 두개의 구별되는 조작을 수행하도록 지시한다.

- 주문자표의 모든 레코드들로부터 성과 이름을 추출하는 SELECT지령
- 결과로 얻은 레코드모임을 새로운 Names표에 넣기 위한 INSERT지령

INSERT ... SELECT와 WHERE문절의 리용

선택적인 WHERE문절은 조건적인 질문을 만들수 있게 한다. 실례로 성이 "김"가인 모든 주문자들에 대한 레코드들을 찾아서 새로운 표에 삽입할수 있다.

```
INSERT INTO Names
SELECT Family_Name, Personal_Name
FROM Customers WHERE Family_Name = '김';
```

2.3.2. UPDATE문

UPDATE지령은 행들의 모임에서 개별적인 렬들의 내용을 변경시키는데 리용된다. UPDATE지령은 보통 갱신하는 행들을 선택하기 위한 WHERE문절과 함께 리용된다.

자료기지응용분야에서 자주 제기되는 요구가 바로 레코드들을 갱신하는것이다. 실례로 주문자가 집을 이사한 경우 그의 주소를 변경해야 한다. 이런 요구를 해결하기 위해서는 특정한 자료들을 선택하는 WHERE문절과 함께 UPDATE문을 리용한다.

```
UPDATE CUSTOMERS
SET District = '평천구역', Dong = '안산2동', Neighbor='13'
WHERE Family_Name = '김' AND Personal_Name = '성철';
```

우의 실례에서는 우선 성과 이름이 일치하는 모든 레코드들을 찾아내기 위하여 WHERE문절을 평가한 다음 이 레코드들에 대한 주소변화를 실시한다.

UPDATE와 계산값들의 리용

렬을 변경하는데 계산식을 리용할수도 있다. 례를 들어 상품명세표에서 재고량을 증가시키는 경우 다음과 같이 UPDATE문과 함께 계산식을 리용할수 있다.

```
UPDATE INVENTORY
SET Qty = Qty + 24
WHERE Name = '대동강텔레비죤';
```

이와 같이 계산되는 UPDATE문을 리용할 때 앞에서 언급된 INSERT와 UPDATE에 대한 규칙들을 지켜야 한다. 특히 계산된 값의 자료형이 변경하려는 마당의 자료형과 같아야 한다.

UPDATE의 유효성판정을 위한 트리거의 리용

SQL언어는 제약조건에 대한 정의와 함께 지적한 조작이 표에 실시될 때 적용되는 보안규칙들을 규정한다. 이 규칙들은 표갱신과 같은 자료기지사건들이 발생할 때 자동적으로 생겨나므로 트리거라고 부르기도 한다.

트리거의 전형적인 리용실례는 상품명세표에 대한 갱신이 정확한가를 판정하는것이다. 다음의 코드는 상품목록표에서 상품의 가격을 15%이상 증가시키려는 경우 자동적으로 취소(되돌리기)하는 트리거를 보여준다.

```
CREATE TRIGGER FifteenPctRule ON INVENTORY FOR INSERT, UPDATE AS
DECLARE @NewCost money
DECLARE @OldCost money
SELECT @NewCost = cost FROM Inserted
SELECT @OldCost = cost FROM Deleted
IF @NewCost > (@OldCost * 1.15)
ROLLBACK Transaction;
```

이 코드에서 리용된 SQL의 ROLLBACK지령은 거래관리지령의 하나이다.

UPDATE와 거래관리지령들의 리용

거래관리는 거래라고 하는 자료기지지령묶음들을 수행하는 관계형자료기지관리체계의 능력에 기인된다. 거래는 순서대로 실행되며 모든 지령들이 성과적으로 완료되여야하는 지령묶음 혹은 지령들의 련속적인 렬이다. 만일 거래기간에 어떤 일이 생기면 자료기지관리체계는 전체 거래를 취소한다. 한편 그것이 성공적으로 완료되면 거래가 자료기지에 보관되거나 위탁될수 있다.

아래의 코드에는 두개의 갱신지령이 있다. 우선 건반의 가격을 305원으로, 다음 마우스의 가격을 215원으로 설정하려 하고있다. 갱신을 시도하기전 원래 건반의 가격이 205원이라면 이 갱신은 명백히 우에서 정의된 15%규칙트리거를 위반한다. 두 갱신이 같은 거래에 포함되여있기때문에 15%규칙트리거에서 ROLLBACK지령이 실행되며 갱신은 실패한다.

```
BEGIN transaction;
UPDATE INVENTORY
   SET Cost = 305
   WHERE Name = 'Keyboard';
UPDATE INVENTORY
   SET Cost = 215
   WHERE Name = 'Mouse';
COMMIT transaction;
```

비록 모든 SQL지령들이 거래내에서 실행되지만 AUTOCOMMIT가 설정되여있지 않는 한 거래 그 자체는 사용자에게 투명하다. 대부분의 자료기지들이 AUTOCOMMIT를 지원하는데 이 추가선택은 개별적인 지령들이 수행되는 차제로 관계형자료기지관리체계에 위탁될수 있게 한다. 이 추가선택은 SET지령과 함께 리용된다.

```
SET AUTOCOMMIT [ON | OFF];
```

자료기지관리체계가 기동될 때 기정으로는 SET AUTOCOMMIT ON지령이 실행된다. 사용자가 거래를 시작할 때 자동위탁이 해소(OFF)되고 그 다음에 거래에서 요구하는 지령들이 발행된다. 거래에서 요구하는 지령들이 모두 정확하게 실행되면 COMMIT지령이 실행될 때 거래가 위탁된다. 거래기간에 어떤 문제든지 일어나면 전체 거래가 ROLLBACK지령에 의해 취소된다.

2.3.3. DELETE지령

DELETE지령은 전체 레코드 혹은 레코드들의 모임을 삭제하는데 리용된다. DELETE지령을 리용할 때 보통 삭제할 레코드들을 추출하는 WHERE문절과 함께 쓰인다. WHERE문절이 없으면 표의 모든 행들이 삭제된다.

```
DELETE FROM Customers
WHERE Family_Name = '김' AND Personal_Name = '성철';
```

우의 SQL문들이 수행되면 주문자료에서 이름이 "김성철"인 주문자의 자료는 전부 지워진다.

제4절. 자료질문언어

자료기지를 응용하는데서 가장 중요한 기능은 어떤 조건을 만족시키는 레코드들을 검색하고 사용자의 요구에 따라 돌려주는것이다.

SQL에서 이러한 능력은 **자료질문언어**(Data Query Language: DQL)에 의하여 제공된다. 형식화된 레코드들을 찾고 돌려주는 과정을 **자료기지질문**이라고 한다.

2.4.1. SELECT문

자료기지질문에서 자료를 돌려주는외에 SELECT문은 UPDATE지령으로 레코드들을 변경하는것과 같은 다양한 조작들에서 자료선택을 위한 다른 SQL지령들과 조합되여 리용될수 있다.

단순한 질문의 기본형식은 돌려주어야 할 렬이름들과 그것들을 찾을수 있는 표 혹은 표들의 이름을 지적하는것이다.

```
SELECT 렬이름1, 렬이름2, ... FROM 표이름;
```

2.4.2. WHERE문절

실천에서 표의 모든 행들을 다 요구하는 경우는 드물다. 대부분의 질문은 어떤 일정한 조건에 맞는 레코드들에서 필요한 마당들만을 요구한다.

이러한 구체적인 질문을 작성하기 위하여 WHERE문절이 리용된다. 아래의 주문자표로부터 남포시에서 살고있는 모든 주문자들의 자료를 요구하는 SQL질문은 다음과 같다.

표 2-3. 주문자표

Family_ Name	Personal_ Name	Sex	State	City	District	Dong
김	성철	남	평양시	평양	중구역	교구동
김	은희	녀	평양시	평양	보통강구역	대보동
김	철	남	함경남도	함흥		사포동
박	성호	남	평안남도	남포	와우도구역	천리마동
강	권혁	남	평양시	평양	중구역	류성동
하	성진	남	황해북도	사리원		경암동
최	영실	녀	강원도	원산		석현동
리	성민	남	평안북도	신의주		역전동
오	성철	남	평양시	평양	평천구역	안산2동

```
SELECT * FROM Customers WHERE City = '남포';
```

이 질문이 수행되면 CITY렬에 《남포》가 있는 모든 행들이 귀환된다. 귀환되는 렬들의 순서는 자료기지에 보관된 순서이지만 행들의 순서는 일정하지 않다.

렬마당들을 지적한 순서대로 검색하게 하려면 SELECT문에서 렬이름들을 요구하는 순서대로 지적한다. 주문자의 성과 이름, 그가 살고있는 동의 순서로 얻으려면 다음과 같이 입력한다.

```
SELECT Family_Name, Personal_Name, Dong FROM Customers
WHERE City = '남포';
```

반대로 동이름을 먼저 출력하고 주문자의 이름을 얻기 위해서는 다음과 같이 질문한다.

```
SELECT Dong, Family_Name, Personal_Name FROM Customers
WHERE City = '남포';
```

주의: 표처리프로그람의 행들과 달리 자료기지표의 레코드들은 절대적인 순서를 가지지 않는다. 사용자가 레코드들의 정렬이 요구되면 SQL의 ORDER BY지령으로 귀환되는 레코드들을 정렬해야 한다.

2.4.3. SQL연산자

지금까지 학습한 질문들은 매우 단순한것들이다. 실천적으로는 다양하게 조합된 여러 마당값들에 기초한 질문들을 리용하게 된다. SQL은 값비교에 기초하여 복합질문들을 작성할수 있게 하는 많은 연산자들을 제공한다.

연산자(operator)는 WHERE문절에서 자료검색이나 변경을 위하여 지적된 조건들을 조합하는 식들에 리용된다. SQL은 여러가지 류형의 연산자들을 가지는데 기본적으로 5가지로 분류할수 있다.

- 비교연산자
- 론리연산자
- 산수연산자
- 모임연산자
- 특수목적연산자

1) 비교연산자

SQL의 연산자들은 WHERE문절에 리용되는 검사조건들을 정의하는데 리용한다. SQL은 아래에 보는바와 같이 표준비교연산자들은 물론 IS NULL과 같은 특수연산자들도 지원한다.

- =
- <>
- >, >=
- <, <=
- IS NULL
- IS NOT NULL

IS NULL과 IS NOT NULL은 지적한 렬이 NULL값을 허용하는가 안하는가를 검사하는데 리용된다.

수값과 문자비교

SQL의 비교연산자들은 수값변수와 문자변수에 다같이 동작한다. 다시 말하여 수값을 비교할 때와 똑같은 방법으로 문자변수를 비교할수 있다는것을 의미한다.

다음의 두 질문은 다같이 유효하다.

```
SELECT * FROM Customers WHERE Sex = '녀' ;
SELECT * FROM Inventory WHERE Item_Number = 1006;
```

만일 CHAR형값이나 VARCHAR형값에 대하여 보다크기 혹은 보다작기연산자를 리용하는 경우 비교는 사전처럼 진행된다. 다음의 질문은 표 2-4와 같은 결과모임을 돌려준다.

```
SELECT *
FROM Customers
WHERE Family_Name >= '리';
```

표 2-4. 사전적인 문자렬비교의 결과

ID	Family_Name	Personal_Name	Sex	State	City	District	Dong
103	박	성호	남	평안남도	남포	와우도구역	천리마동
105	하	성진	남	황해북도	사리원	<NULL>	경암동
106	최	영실	녀	강원도	원산	<NULL>	석현동
107	리	성민	남	평안북도	신의주	<NULL>	역전동

IS NULL연산자

NULL값은 자료가 없다는것을 표시하며 따라서 비교연산자로 평가할수 없다. SQL은 NULL에 대한 검사를 위해 IS NULL과 IS NOT NULL연산자를 제공한다. 실례로 주문자표에 전화번호렬을 추가하는 경우 주문자에게 전화가 없다면 그 마당값을 NULL로 남겨둔다. 전화번호가 없는 주문자들을 얻는 질문은 다음과 같이 만들수 있다.

```
SELECT * FROM Customers WHERE Phone IS NULL;
```

LIKE와 NOT LIKE연산자

비교연산자뿐만아니라 SQL에서는 CHAR형변수와 VARCHAR형변수안에 있는 부분문자렬에 대한 검사를 진행하는 전용연산자를 제공한다.

- LIKE
- NOT LIKE

LIKE연산자와 그의 부정연산자인 NOT LIKE연산자는 통용문자와 조합되여 문자렬비교에서 매우 강력한 도구로 리용될수 있다. 통용문자는 다음과 같다.

- 밑줄(_): 한개문자에 대한 통용문자
- 퍼센트(%): 여러문자에 대한 통용문자

례를 들어 주문자표에서 성의 첫글자가 《김》으로 시작되는 모든 주문자레코드들을 찾기 위한 질문은 다음과 같이 쓸수 있다.

```
SELECT * FROM Customers WHERE First_Name LIKE '김_';
```

NOT LIKE도 LIKE와 류사한 방식으로 동작한다. 례를 들어 도시의 첫 글자가 《평》으로 시작되지 않는 도시에서 살고있는 모든 주문자들을 찾기 위한 질문은 다음과 같이 NOT LIKE를 리용할수 있다.

```
SELECT * FROM Customers WHERE City NOT LIKE '평%';
```

련결연산자의 리용

련결연산자(+ 혹은 ||)는 어떤 문자렬에 다른 문자렬을 덧붙이는데 리용된다. 례를 들어 성과 이름을 련결하여 완전한 이름을 얻는 질문은 다음과 같다.

```
SELECT Family_Name + Personal_Name AS NAME FROM Customers;
```

2) 론리연산자

WHERE문절에서는 둘 혹은 그 이상의 비교연산을 조합할 필요가 많이 제기된다. SQL에서는 비교연산자들을 결합하기 위한 다음의 표준론리연산자들을 제공한다.

- AND
- OR
- NOT

AND연산자의 리용

AND연산자는 결합하는 비교연산들중 매 결과가 모두 참(true)일 때에만 참으로 평가되는 비교연산에 리용된다. 어느 하나라도 참이 아니면 AND연산결과는 거짓(false)이다. 례를 들어 주문자표에서 평양시에 사는 성이 《김》가인 주문자들에 대한 자료를 얻기 위한 질문은 다음과 같다.

```
SELECT *
FROM Customers WHERE Family_Name = '김' AND State = '평양시';
```

OR연산자의 리용

OR연산자는 결합하는 비교연산들중에서 어느 하나라도 참이면 그 결과가 참으로 되는 비교연산에 리용된다. 례를 들어 남포시 혹은 평안남도에서 사는 주문자들을 모두 찾으려 할 때 OR연산자를 리용한다.

```
SELECT * FROM Customers WHERE City = '남포' OR State = '평안남도';
```

NOT연산자의 리용

NOT연산자는 비교결과를 반전하는데 리용된다. 례를 들어 평양시나 평안남도에서 살지 않는 성이 《김》가인 모든 주문자들을 찾는 질문은 다음과 같다.

```
SELECT * FROM Customers
WHERE Family_Name ='김' AND NOT (State ='평양시' OR State ='평안남도');
```

우에서 보는것처럼 괄호를 리용하여 론리연산자들을 결합할수 있다. 이때 연산순서는 괄호안에 있는 연산들이 먼저 수행된다.

3) 산수연산자

SQL은 더하기(+), 덜기(-), 곱하기(*), 나누기(/)와 같은 일반적인 산수연산자들과 옹근수나누기의 나머지를 돌려주는 나머지연산자(%)를 지원한다.

WHERE문절에서 산수연산자의 리용

산수연산자는 WHERE문절에서 가장 많이 리용된다. 례를 들어 다음의 문은 표 2-5에서 소매가격이 10.00아래인 상품항목들을 보기 위한 질문이다. 소매가격을 얻기 위해 원가에 판매부과률를 곱하는 계산을 WHERE문절에서 하고있다.

표 2-5. 상품목록표

Item_Number	Name	Description	Qty	Cost
1001	고려인삼술	주류	178	1.95
1002	평양술	주류	97	1.87
1003	백두산들쭉술	주류	103	2.05
1004	강계인풍술	주류	15	0.98
1005	대동강맥주	청량음료	217	0.26

Item_Number	Name	Description	Qty	Cost
1006	평양맥주	청량음료	162	0.17
1007	랭천사이다	청량음료	276	0.13
1008	모란과자	당과류	144	0.31
1009	살구씨향과자	당과류	96	0.47
1010	박하사탕	당과류	84	0.25

```
SELECT Item_Number, Name, Description, Cost, Cost * 1.6 AS Retail
FROM Inventory
WHERE Cost * 1.6 < 10;
```

계산된 결과렬의 작성

산수연산자들은 계산결과마당을 작성하는데도 쓸모가 있다. 실례로 다음과 같이 원가로부터 소매가격을 계산할수 있다.

```
SELECT Item_Number, Name, Description, Cost, Cost * 1.6 AS Retail
FROM Inventory
```

이 질문은 표 2-6에 보여준것처럼 "Retail"이라는 추가렬을 함께 돌려준다.

표 2-6. 계산된 결과표

Item_Number	Name	Description	Cost	Retail
1001	고려인삼술	주류	1.95	3.12
1002	평양술	주류	1.87	2.992
1003	백두산들쭉술	주류	2.05	3.279
1004	강계인풍술	주류	0.98	1.568
1005	대동강맥주	청량음료	0.26	0.416
1006	평양맥주	청량음료	0.17	0.272
1007	랭천사이다	청량음료	0.13	0.208
1008	모란과자	당과류	0.31	0.496
1009	살구씨향과자	당과류	0.47	0.752
1010	박하사탕	당과류	0.25	0.4

❖ 별명의 리용 ❖

앞 실례에서 키단어 AS가 지령에 리용되였다. 선택적인 AS문절은 식에 의미있는 이름을 할당할수 있게 한다. 별명은 다른 문에서 그 렬을 지적할 필요가 있을 때 보통의 렬이름으로 리용할수 있다. 이 실례에서 AS는 계산된 값렬에 이름 혹은 별명(실례에서는 Retail)을 할당한다.
별명을 할당하고 리용할 때 한 문절의 출력은 다음 문절의 입력으로 되기때문에 SQL이 다양한 문절들을 처리하는 순서에 마음을 써야 한다. SQL지령의 부문절(subclauses)들이 처리되는 순서를 다음 목록에 보여주었다.

- FROM문절
- WHERE문절
- GROUP문절
- HAVING문절
- SELECT문절
- ORDER BY문절

앞 실례에서는 별명을 할당하기 위한 키단어 AS가 SELECT문절에서 리용되였기때문에 그 별명을 WHERE문절에서는 리용할수 없다. 왜냐하면 WHERE문절이 먼저 수행된 다음에야 SELECT문이 실행되기때문이다. 그러나 그 별명을 다음 실례와 같이 ORDER BY문절에서는 리용할수 있다.

```
SELECT Item_Number, Name, Description, Cost, Cost * 1.6 AS Retail
FROM Inventory ORDER BY Retail;
```

4) 모임연산자

모임연산자(set operator)는 각이한 질문에 의한 결과들을 단일한 결과모임으로 결합할수 있게 한다. 기본적인 모임연산자들은 다음과 같다.

- UNION과 UNION ALL : 두 질문의 결과들을 결합하여 돌려준다.
- INTERSECT : 두 질문에 다 만족되는 행들만 돌려준다.
- EXCEPT : 첫번째 질문결과에서 두번째 질문에 없는 행들만 돌려준다.

UNION과 UNION ALL의 리용

UNION ALL은 두 질문의 결과들을 모두 합쳐서 돌려주며 UNION은 두 질문결과들중에서 중복을 제거한다. 례를 들어 평양시에 사는 《김》가성을 가진 주문자들에 대한 질문과 《박》가성을 가진 모든 주문자들에 대한 질문을 결합하기 위해 UNION을 리용할수 있다.

```
SELECT *
FROM Customers
WHERE Family_Name = '김' AND City = '평양'
```

```
UNION
SELECT *
FROM Customers
WHERE Family_Name = '박';
```

INTERSECT와 EXCEPT

이 두 연산자들도 UNION연산자와 같은 문장구성법을 따른다. 이 연산자들을 사용하려면 사전에 자료기지관리체계가 그것들을 지원하고있는가를 확인해보아야 한다.

5) 특수목적연산자들

SQL은 특수한 기능들을 수행하는 많은 연산자들을 제공한다. 다른 언어들에서는 대체로 이러한 기능들을 수행하는 코드를 특별히 작성해야 할것이다. SQL은 수속형언어가 아니므로 이것들은 아주 쓸모있는 기능들이다.

IN연산자

IN연산자는 목록과 대조하여 마당들을 비교하는 강력한 수단이다. 례를 들어 평양시와 남포시에 있는 주문자들을 찾으려면 다음 질문을 리용할수 있다.

```
SELECT *
FROM Customers
WHERE City IN ('평양', '남포');
```

IN연산자는 수값들에도 작용한다. 상품목록표에서 Item_Number에 따라 항목들을 선택하는 경우에는 다음의 질문을 리용할수 있다.

```
SELECT *
FROM Inventory
WHERE Item_Number IN (1001, 1003, 1004);
```

BETWEEN연산자

BETWEEN연산자는 일정한 범위안에 있는 값들을 가진 마당들을 선택하는데 리용된다. 다음의 질문은 원가가 1.03부터 1.95범위에 있는 항목들을 찾아 돌려준다.

```
SELECT *
FROM inventory
WHERE Cost BETWEEN 1.03 AND 1.95;
```

DISTINCT연산자

DISTINCT연산자는 질문에 맞는 레코드들을 모두 돌려주는 SELECT문과는 달리 중복된 레코드는 제거할수 있게 한다. 실례로 주문자표에서 중복없이 주문이 들어온 도들을 모두 보려면 SELECT문과 DISTINCT연산자를 함께 리용한다.

```
SELECT DISTINCT State
FROM Customers;
```

TOP연산자

TOP연산자는 질문결과모임으로부터 첫 n개의 행 혹은 첫행부터 n%에 해당한 행들만 출력되게 할수 있다. 퍼센트로 지적할 때 n은 0과 100사이의 옹근수여야 한다.

아래의 질문은 Inventory표에서 첫행부터 25%에 해당한 행들을 돌려준다.

```
SELECT TOP 25 PERCENT * FROM Inventory;
```

이 질문의 결과모임을 표 2-7에 보여주었다.

표 2-7. 25%에 해당한 레코드들

Item_Number	Name	Description	Qty	Cost
1001	고려인삼술	주류	178	1.95
1002	평양술	주류	97	1.87
1003	백두산들쭉술	주류	103	2.05

만일 질문에 ORDER BY문절이 있는 경우에는 ORDER BY문절에 의해 배렬된 첫 n개의 행(혹은 n%의 행)들을 출력한다.

❖ 탈출문자렬(Escape Sequences) ❖

탈출문자렬은 SQL에서 특별한 의미를 가지는 어떤 문자를 사용자가 다른 용도로 사용하려는 경우에 리용된다. 전형적인 실례는 웃반점(')의 리용이다. SQL에서 웃반점은 단일인용부호이기때문에 SQL은 그것을 CHAR 혹은 VARCHAR형 변수의 끝으로 읽으며 문자렬의 나머지부분이 취급되려 할 때 SQL오유를 발생시킨다.

이 문제의 해결은 간단하다. 단순히 웃반점을 2중으로('') 겹쳐놓는다.

탈출문자렬을 요구하는 다른 두개의 문자가 있다.

  % (퍼센트)
  _ (밑줄)

이것들은 질문의 마지막에 탈출문자를 정의함으로써 조종할수 있다. 탈출문자는 열쇠단

어 escape를 리용하여 대괄호({})안에 정의된다.

```
 {escape '탈출문자'}
```

례를 들어 다음의 질문은 표에서 밑줄로부터 시작되는 이름들을 찾는다. 여기서는 역사선(\)문자를 탈출문자로 리용하였다.

```
SELECT Name
FROM Variables
WHERE ID LIKE '\_%' {escape '\'};
```

2.4.4. 보조질문의 리용

다른 SQL문의 보조적인 부분으로 리용되는 간단한 질문을 **보조질문**(subquery)이라고 한다. 보조질문은 다음과 같은 SQL문들속에 포함될수 있다.

- SELECT 혹은 SELECT ... INTO
- INSERT ... INTO
- DELETE
- UPDATE
- 다른 질문이나 보조질문의 내부

보조질문들은 다른 SQL문에서 조작하여야 할 중간결과를 제공한다. 이 경우 보조질문은 기본질문의 WHERE 혹은 HAVING문절에서 평가되여야 할 값들의 모임을 제공하기 위하여 SELECT문을 리용한다.

보조질문은 WHERE 혹은 HAVING문절에서 다음과 같은 비교 및 표현식들의 오른쪽에 리용된다.

- ANY, ALL 혹은 SOME을 리용한 비교
- IN 혹은 NOT IN을 리용한 표현식
- EXISTS 혹은 NOT EXISTS를 리용한 표현식

1) ANY, SOME, ALL술어부

많은 경우 보조질문은 하나이상의 값들을 돌려주기때문에 비교하기전에 보조질문의 결과들을 조종하기 위한 특별한 술어부를 요구한다. 례를 들어 상품명세항목들중 당과류보다 가격이 비싼 항목들을 찾으려는 경우 우선 당과류에 대한 모든 가격행들을 얻기 위한 보조질문이 요구된다.

```
(SELECT Cost FROM Inventory
WHERE Description = '당과류');
```

보조질문을 작성할 때 전체 보조질문은 괄호로 닫겨있어야 한다. ANY연산자나 SOME연산자는 동의어로서 다같이 보조질문에서 검색한 임의의 레코드들과 비교조건을 만족시키는 행들을 검색하는데 리용된다.

```
SELECT * FROM Inventory
WHERE Cost >= ANY
   (SELECT Cost FROM Inventory
    WHERE Description = '당과류');
```

이 질문은 상품명세표에서 제일 눅은 당과류와 가격이 같거나 비싼 모든 상품항목들을 돌려준다.

Item_Number	Name	Description	Qty	Cost
1001	고려인삼술	주류	178	1.95
1002	평양술	주류	97	1.87
1003	백두산들쭉술	주류	103	2.05
1004	강계인풍술	주류	15	0.98
1005	대동강맥주	청량음료	217	0.26
1008	모란과자	당과류	144	0.31
1009	살구씨향과자	당과류	96	0.47
1010	박하사탕	당과류	84	0.25

ALL술어부는 보조질문에서 검색한 모든 레코드들과의 비교를 만족시키는 레코드들을 기본질문에서 검색하는데 리용된다. 앞의 실례에서 ANY를 ALL로 바꾸면 질문은 다음과 같이 모든 당과류들중 제일 비싼 당과류와 가격이 같거나 더 비싼 상품항목들을 돌려준다.

Item_Number	Name	Description	Qty	Cost
1001	고려인삼술	주류	178	1.95
1002	평양술	주류	97	1.87
1003	백두산들쭉술	주류	103	2.05
1004	강계인풍술	주류	15	0.98
1009	살구씨향과자	당과류	96	0.47

2) IN과 NOT IN술어부

IN술어부는 목록에 대비하여 값들을 비교하는데 리용된다. 다음의 례는 평양시와 평안남도에 있는 모든 주문자들을 찾기 위하여 주문자들의 도를《평양시》와《평안남도》를 포함하는 목록과 대비하여 검사한다.

```
SELECT * FROM Customers
WHERE State IN ('평양시','평안남도')
```

또한 대비할 목록을 만들기 위해 IN술어부안에 보조질문을 쓸수도 있다. 다음의 코드는 보조질문을 리용하여 주문항목표(Ordered_Items)에서 주문번호(Order_Number)가 2인 상품항목들의 번호목록을 만들고 그와 관련된 상품명세자료를 얻기 위해 IN술어부를 리용하고있다.

```
SELECT *
FROM Inventory
WHERE Item_Number IN
   (SELECT Item_Number
    FROM Ordered_Items
    WHERE Order_Number = 2);
```

이 질문이 돌려주는 결과모임은 다음과 같다.

Item_Number	Name	Description	Qty	Cost
1001	고려인삼술	주류	178	1.95
1004	강계인풍술	주류	15	0.98
1005	대동강맥주	청량음료	217	0.26
1010	박하사탕	당과류	84	0.25

IN술어부를 NOT연산자와 함께 리용하면 선택목록에 포함되지 않는 모든 항목들을 선택할수 있다.

3) EXISTS와 NOT EXISTS술어부

EXISTS와 NOT EXISTS는 참 혹은 거짓을 돌려주는 술어부이다. 이 술어부들은 보조질문이 임의의 레코드들을 돌려주는가 아닌가를 결정하는《참/거짓》비교에 리용된다. 례를 들어 어떤 상품을 주문한 주문자의 주문자ID와 주문번호가 일치하는 주문상품항목들의 결과모임을 얻기 위해 보조질문을 리용할수 있다. 다음 아래에 보여준것처럼

EXISTS술어부와 함께 《Description ='청량음료'》비교를 리용하여 그 사람이 주문한 청량음료의 종류를 찾을수 있다.

```
SELECT DISTINCT Name FROM Inventory
WHERE Description = '청량음료' AND EXISTS
   (SELECT *
    FROM Customers c, Ordered_items oi, Orders o, Inventory i
    WHERE c.Customer_ID = o.Customer_ID AND
          oi.Order_Number = o.Order_Number AND
          oi.Item_Number = i.Item_Number);
```

이 질문은 다음의 결과모임을 돌려준다.

Name
대동강맥주
랭천사이다
평양맥주

보조질문의 SELECT목록에 있는 《*》의 리용을 주의해보기 바란다. EXISTS는 true 혹은 false만을 돌려주기때문에 보다 구체적인 지적으로 더 이상 얻을것이 없다. 때문에 EXISTS술어부는 《*》와 함께 쓰는것이 좋다.

주의: EXISTS술어부는 맞는것을 하나라도 찾으면 탐색을 멈춘다. 따라서 대응하는 다른 행들을 계속 찾아내는 질문보다 더 빠르고 효과적이다.

4) 보조질문의 추가적 리용

SELECT지령에서 보조질문의 리용

SELECT문절을 리용한 질문에서 단순한 자료마당값대신 계산되는 값을 리용할수 있은것처럼 보조질문들이 돌려주는 결과를 리용할수 있다. 이러한 기능은 류사한 상품들의 평균가격과 같이 표로부터 추출한 다른 값과 상품들의 가격을 비교하는 경우에 쓸모가 있다.

```
SELECT Name, Cost,
(SELECT AVG(Cost) FROM Inventory WHERE Description = '당과류') AS
'AVERAGE'
FROM Inventory WHERE Description = '당과류';
```

이 질문의 결과는 다음과 같다.

NAME	COST	AVERAGE
모란과자	0.31	0.34
살구씨향과자	0.47	0.34
박하사탕	0.25	0.34

INSERT지령과 보조질문

어떤 표에서 선택한 레코드들을 다른 표에 삽입하는 실례에서 보조질문의 리용을 보기로 한다. 다음의 실례에서는 보조질문을 리용하여 황해북도에서 살고있는 주문자들의 주문자ID를 선택하고있다. 다음 선택된 주문자ID를 가진 주문자들에 대한 일부 자료들을 선택하여 Employees표에 삽입한다.

```
INSERT INTO Employees (Employee_ID, Family_Name, Personal_Name)
   SELECT Customer_ID, Family_Name, Personal_Name
      FROM Customers
      WHERE Customer_ID IN
         (SELECT Customer_ID
            FROM Customers
            WHERE State = '황해북도');
```

UPDATE지령과 보조질문

UPDATE지령과 보조질문을 함께 리용하는 경우도 많다. 다음 실례에서는 보조질문을 리용하여 주문자표에서 갱신되여야 할 주문자ID를 선택한 다음 UPDATE지령의 WHERE문절에서 그 주문자ID를 리용한다.

```
UPDATE Employees
   SET Personal_Name = '은희'
      WHERE Employee_ID IN
         (SELECT Customer_ID
          FROM Customers
          WHERE Personal_Name = '성진');
```

갱신지령에서 보조질문리용의 한가지 우점은 정확한 자료모임을 얻었는가를 확인하기 위하여 자체로 보조질문을 쉽게 검사할수 있다는것이다. 다음 그것이 틀림없다고 확인되면 그 보조질문을 실제적인 갱신지령속에 넣을수 있다.

DELETE지령과 보조질문

마지막으로 DELETE지령과 보조질문의 리용을 보기로 하자.

```
DELETE FROM Customers
WHERE Customer_ID IN (SELECT Employee_ID FROM Employees);
```

보조질문의 중첩

질문안에 보조질문을 리용할수 있는것처럼 보조질문안에서 또 다른 보조질문을 리용할수 있다. 보조질문중첩을 위한 문장구성법은 다음과 같다.

```
SELECT *
FROM Tables
WHERE
   (SUBQUERY
     (SUBQUERY
       (SUBQUERY)));
```

호상련관된 보조질문

지금까지 토론된 대부분의 보조질문들은 보조질문 그자체내에 정의된 표들에만 문의하는 자립적인것들이였다. 보조질문의 이러한 자립적인 특징은 독립적인 질문으로서 그것들을 쉽게 검사할수 있게 한다. 그러나 때때로 보조질문에서 외부참조의 리용이 쓸모있다.

호상련관된 보조질문(correlated subquery)이란 외부질문의 값에 의존하는 보조질문들이다. 보조질문에서 외부질문에 있는 표에 대한 참조를 **호상련관된 참조**(correlated reference)라고 한다. 다음의 실례는 CUSTOMERS, INVENTORY, ORDERS표에 대한 참조에서 호상련관된 질문을 보여주고있다. 이 표들은 외부질문의 From문절에는 나타나있지만 보조질문의 From문절에는 없다.

```
SELECT c.Family_Name, c.Personal_Name, o.Order_Number, i.Item_Number,
i.Name
FROM Customers c, Inventory i, Orders o
WHERE i.Description = '청량음료' AND EXISTS
  (SELECT *
      FROM Ordered_Items oi
      WHERE c.Customer_ID = o.Customer_ID AND
            oi.Order_Number = o.Order_Number AND
            oi.Item_Number = i.Item_Number);
```

이 실례에서 보조질문이 접근하는 대부분의 표들은 기본질문에서 정의되였다. 이 질문은 다음과 같은 결과모임을 돌려준다.

Family_Name	Personal_Name	Order_Number	Item_Number	Name
김	은희	2	1005	대동강맥주
김	철	3	1006	평양맥주
김	성철	5	1006	평양맥주
김	성철	5	1007	랭천사이다

호상련관된 질문들은 외부질문에서 식별된 표의 매개 행에 대하여 한번씩 반복적으로 실행되기때문에 매우 비효률적일수 있다.

2.4.5. 질문결과의 정렬

자료기지에서 자료를 검색할 때 일반적으로 제기되는 요구는 하나이상의 렬들을 자모순 혹은 수값순서로 정렬하는것이다. 결과모임에 대한 정렬은 다음과 같이 ORDER BY 문절을 리용한다.

```
SELECT Family_Name, Personal_Name, City, State
FROM Customers
WHERE Family_Name = '김'
ORDER BY Personal_Name;
```

이 질문의 결과는 다음과 같다.

표 2-8. ORDER BY에 의하여 정렬한 결과

Family_Name	Personal_Name	City	State
김	성철	평양	평양시
김	은희	평양	평양시
김	철	함흥	함경남도

다음과 같이 열쇠단어 DESC를 추가하면 내림순으로 변화시킬수 있다.

```
SELECT *
FROM Customers
WHERE Family_Name = '김'
ORDER BY Personal_Name DESC;
```

여러개 렬의 정렬은 정렬목록을 리용한다. 실레로 Family_Name에 의하여 오름순서로 자료를 정렬한 다음 Personal_Name에 따라 내림순으로 다시 정렬하는 SQL문은 다음과 같다.

```
SELECT Family_Name, Personal_Name, State, City
FROM Customers
ORDER BY Family_Name, Personal_Name DESC;
```

ORDER BY를 리용하는데서 규칙은 다음과 같다.

- ORDER BY문절은 SELECT문에서 맨 마지막에 넣어야 한다.
- 기정으로 정의된 정렬순서는 오름순이다.
- 열쇠단어 ASC를 리용하면 오름순으로 정렬된다.
- 열쇠단어 DESC를 리용하면 내림순으로 정렬된다.
- ORDER BY문절에서 렬이름이나 표현식을 리용할수 있다.
- ORDER BY문절에 있는 렬이름들은 선택목록에서 꼭 지적하지 않아도 된다.
- NULL값은 보통 정렬순서에서 먼저 나타난다.

2.4.6. 질문결과의 요약

질문결과에 대한 또 하나의 공통적인 요구는 질문이 돌려주는 자료를 임의의 방식으로 요약할수 있도록 다양한 그룹들로 분해하는것이다. GROUP BY문절은 레코드묶음에 대한 평균 혹은 합계와 같은 계산들을 수행할수 있도록 자료기지의 레코드들을 결합할수 있게 한다.

GROUP BY문절은 다음 실레에 보여준것처럼 지적한 마당에서 동일한 값들을 가진 레코드들을 하나의 레코드로 결합한다.

```
SELECT Description, COUNT(Description) AS 'Count',
       AVG(Cost) AS 'Average Cost'
FROM Inventory
GROUP BY Description;
```

이 질문의 결과는 다음과 같다.

Description	Count	Average Cost
주류	4	1.713
청량음료	3	0.187
당과류	3	0.343

GROUP BY문절은 어떤 렬에서 같은 값을 가진 모든 레코드들을 하나의 레코드로 결합하기때문에 SELECT문절에 렬거된 매개 렬이름들은 GROUP BY문절에 지적된 렬이든지 혹은 COUNT()나 AVG()와 같은 렬함수이여야 한다. 이것은 이름에 따라 개별적인 주문자들의 목록을 선택하고 GROUP BY를 리용하여 그것들을 한 그룹으로 계산할수는 없다는 것을 의미한다. 그러나 ORDER BY문절에서 하나이상의 렬들을 리용할수 있은것과 같이 하나이상의 렬들을 그룹화할수 있다.

SELECT문에서 지적된 모든 렬이름은 GROUP BY문절에서도 언급되여야 한다. 어느 쪽에도 언급되지 않은 렬이름은 오유로 된다. GROUP BY문절은 GROUP BY문절속의 매개 유일한 렬조합에 대하여 한개의 행을 돌려준다.

1) 집계함수

집계함수(aggregate function)은 어떤 렬의 자료에 대한 연산을 진행하여 하나의 결과값을 돌려준다. 이것은 개별적인 자료요소들에 대하여 작용하는 산수연산자, 론리연산자, 문자연산자들과는 다르다.

관계형자료기지관리체계는 대체로 다음과 같은 집계함수를 가진다.

- SUM - 렬값들의 합
- AVG - 렬값들의 평균값
- STDEV - 렬값들의 표준편차
- COUNT - 렬에서 행들의 총수
- MAX - 렬에서 최대값
- MIN - 렬에서 최소값

집계함수는 자료요소의 그룹에 대한 통계적 혹은 요약정보를 제공하는데 리용된다. 이 그룹들은 GROUP BY문절에 의해 명확히 작성될수도 있고 집계함수들이 전체 결과모임인 기정의 그룹에 적용될수도 있다. 아래에 가장 일반적인 집계함수들의 리용실례를 보여주었다.

```
SELECT Description, COUNT(Description) AS 'Count',
   AVG(Cost) AS 'Average Cost', MIN(Cost) AS 'Lowest Cost',
   MAX(Cost) AS 'Highest Cost'
FROM Inventory
GROUP BY Description;
```

이 질문들은 다음의 결과들을 생성한다.

Description	Count	Average Cost	Lowest Cost	Highest Cost
주류	4	1.713	0.98	2.05
청량음료	3	0.187	0.13	0.26
당과류	3	0.343	0.25	0.47

2) 그룹려과를 위한 HAVING문절의 리용

WHERE문절을 리용하여 레코드들을 려과한것과 류사한 방법으로 그룹 그 자체를 려과할수 있다. 례를 들어 판매액을 도별로 분석하되 주문자수가 작은 도들은 무시할수 있다. 그룹들을 려과하기 위해서는 GROUP BY문절 다음에 HAVING문절을 적용한다. HAVING문절은 그룹들에 대한 제한조건을 규정함으로써 자료기지관리체계가 그 조건을 만족시키는 그룹들에 대한 결과만을 돌려주게 한다. 다음의 실례는 도별로 주문자들의 총수를 계산하고 주문자가 한명뿐인 도들은 제외하기 위하여 HAVING문절을 리용하고있다.

```
SELECT Description, State, COUNT(State) AS 'Count',
   SUM(oi.QTY * i.Cost) AS Total
FROM Customers c, Orders o, Ordered_Items oi, Inventory i
WHERE C.Customer_ID = o.Customer_ID AND
   o.Order_Number =oi.Order_Number AND
   i.Item_Number = oi.Item_Number
GROUP BY STATE, DESCRIPTION
HAVING COUNT(State) > 1;
```

이 질문의 결과는 다음과 같다.

Description	State	Count	Total
주류	평양시	2	4.88
청량음료	평안남도	2	0.64

HAVING문절에는 AND와 OR로 여러개의 술어부들을 결합할수 있다. 매 술어부는 (COUNT(State)와 같은) 그룹의 속성을 그룹의 다른 속성이나 상수와 비교한다.

우에서 본 실례에서 GROUP BY문절을 빼놓음으로써 전체 결과모임에 HAVING문절을 적용할수도 있다. 이 경우 자료기지관리체계는 전체 표를 하나의 그룹으로 취급하며 따라서 기껏해서 하나의 결과행이 주어진다. HAVING조건이 전체표에 대하여 참이 아니면 어떤 결과행도 돌아오지 않는다.

3) SQL질문의 효률성을 개선하기 위한 색인의 리용

색인을 리용하면 자료기지의 성능을 대단히 개선할수 있다. 색인은 표나 뷰에서 어떤 특정한 항목을 빨리 찾아볼수 있게 한다. 사실상 색인은 표나 뷰에서 행들에 대한 지적자들의 순서화된 배렬이다.

사용자가 매 행에 열쇠로서 유일한 ID를 할당할 때에는 이미 그 표에 대한 색인을 미리 정하고있다. 이것은 ID렬에 기초하여 표들을 결합할 때 필요한 ID에 의한 항목찾기를 자료기지관리체계가 훨씬 빨리 할수 있게 한다.

SQL의 CREATE INDEX문은 임의의 렬이나 렬들의 그룹에 색인을 추가할수 있게 한다. 례를 들어 주문자이름에 의한 탐색이 요구될 때 그 표가 기본열쇠에 대한 내장된 색인을 가진다는 사실은 전혀 도움이 되지 않으며 따라서 자료기지관리체계는 질문에 부합되는 모든 주문자들의 이름을 찾기 위하여 전체 표에 대한 무지한 탐색을 진행하지 않으면 안된다. 만일 주문자이름에 의해 질문을 많이 할 계획이라면 주문자이름렬에 대하여 색인을 추가해야 한다.

색인을 추가하기 위한 SQL지령은 CREATE INDEX열쇠단어 다음에 색인이름을 지적하고 색인할 표이름과 렬목록을 정의한다. 아래에 그 실례를 준다.

```
CREATE INDEX STATE_INDEX ON CUSTOMERS(State);
```

색인을 제거하기 위한 DROP INDEX지령은 다음과 같다.

```
DROP INDEX CUSTOMERS.STATE_INDEX;
```

❖ SQL지령의 형식화 ❖

SQL엔진은 여러개의 공백을 무시하기때문에 명백한 서술을 위하여 행바꾸기(LINE BREAK)들을 잘 삽입하여야 한다. 습관적으로 FROM문절, WHERE문절과 같은 기본 문절들은 대체로 새행에 쓰는것이 보통이다. 지령을 형식화하는 좋은 방법은 읽기 쉽게 하는것이다.

열쇠단어, 표이름, 렬이름들은 대소문자를 구별하지 않지만 표안에서 레코드들의 내용은 대소문자를 구별한다. 이것은 SQL문을 보다 읽기 쉽게 하기 위해 대문자를 리용할수 있다는것을 의미한다.

2.4.7. 여러 표들로부터 자료의 결합

자료기지들에서 정보는 론리적으로 련관된 자료모임들을 포함하는 여러개의 각이한 표들에 분포되여있다.

앞에서는 다음 4개의 표들을 포함하는 전형적인 자료기지를 실례로 하였다.

- Customers표- 주문자ID, 주문자이름, 주문자의 주소를 포함한다.
- Inventory표 - 상품번호, 상품이름과 종류, 원가와 수량을 포함한다.
- Orders표 - 주문번호, 주문자ID, 주문날자와 수송날자를 포함한다.
- Ordered_Items표 - 주문번호, 상품번호와 수량을 포함한다.

주문자가 어떤 상품을 주문하면 주문번호를 할당하고 주문자ID와 주문날자를 포함하는 Orders표에 등록한다. 다음 주문번호, 상품번호, 수량을 기록하는 Ordered_Items표에 항목들이 기입된다. 이 표들을 표 2-9부터 2-12까지 보여주었다.

표 2-9. 주문자표(Customers)

Cus-tomer_ID	Family_Name	Personal_Name	Sex	State	City	District	Dong	Neighbor
100	김	성철	남	평양시	평양	중구역	교구동	32
101	김	은희	녀	평양시	평양	보통강구역	대보동	17
102	김	철	남	함경남도	함흥		사포동	23
103	박	성호	남	평안남도	남포	와우도구역	천리마동	65
104	강	권혁	남	평양시	평양	중구역	류성동	23
105	하	성진	남	황해북도	사리원		경암동	28
106	최	영실	녀	강원도	원산		석현동	26
107	리	성민	남	평안북도	신의주		역전동	33
108	오	성철	남	평양시	평양	평천구역	안산2동	56

표 2-10. 상품목록표(Inventory)

Item_Number	Name	Description	Qty	Cost
1001	고려인삼술	주류	178	1.95
1002	평양술	주류	97	1.87
1003	백두산들쭉술	주류	103	2.05
1004	인풍술	주류	15	0.98
1005	대동강맥주	청량음료	217	0.26
1006	평양맥주	청량음료	162	0.17
1007	랭천사이다	청량음료	276	0.13
1008	모란과자	당과류	144	0.31
1009	살구씨향과자	당과류	96	0.47
1010	박하사탕	당과류	84	0.25

표 2-11. 주문표(Orders)

Order_Number	Customer_ID	Order_Date	Ship_Date
2	101	2007-04-05	2007-04-07
3	102	2007-04-09	2007-04-11
4	104	2007-04-09	2007-04-11
5	100	2007-04-10	2007-04-12
6	107	2007-04-12	2007-04-14
7	106	2007-04-14	2007-04-16

표 2-12. 주문항목표(Ordered_Items)

ID	Order_Number	Item_Number	Qty
5000	2	1001	2
5001	2	1004	1
5002	2	1005	4
5003	2	1010	6
5004	3	1006	4
5005	3	1009	2
5006	4	1002	5
5007	4	1003	2
5008	5	1006	3
5009	5	1007	1
5010	5	1008	2

1) JOIN에 의한 표결합

SQL의 가장 강력한 기능의 하나가 JOIN문을 리용하여 여러개의 표들로부터 자료를 결합하는 능력이다. JOIN을 리용하여 주문자이름, 수송주소 그리고 주문된 항목들의 종류, 수량, 단가, 가격과 같은 청구정보를 보여주는 구체화된 계산서를 만들수 있다.

JOIN에서 열쇠의 리용

SQL의 JOIN을 리해하는데서 가장 중요한것은 기본열쇠와 외부열쇠의 리용이다. 우의 실례에서 4개의 표들은 각각 주문자ID 혹은 상품번호와 같은 식별자를 가진다. 이 식별자들은 표의 기본열쇠이며 주어진 행에 대한 유일한 참조를 제공한다.

표에서 외부열쇠는 다른 표에서 기본열쇠로 리용되는 렬이다. 례를 들어 주문표는

주문표의 기본열쇠로 Order_Number렬을 가지며 주문자표의 기본열쇠인 Customer_ID렬을 외부열쇠로 포함하고있다.

갈기결합에 의한 여러표 자료의 접근

SQL결합들은 열쇠를 비교하여 각이한 표들에서 대등한 렬들을 맞추는 방법으로 진행된다. 결합의 가장 일반적인 류형은 어떤 표에서 다른 표와 같은 항목번호를 가지는 항목들을 찾는 갈기결합(Equi-Joins)이다.

JOIN문을 작성하는데는 두가지 방법이 있다. 첫번째는 열쇠단어 JOIN을 지적하는 방법이다.

```
SELECT Family_Name, Personal_Name, Order_Number
FROM Customers c INNER JOIN Orders o
   ON c.Customer_ID = o.Customer_ID;
```

두번째는 JOIN이 없이 WHERE문절을 리용하는 형식이다.

```
SELECT Family_Name, Personal_Name, Order_Number
FROM Customers c, Orders o
   WHERE c.Customer_ID = o.Customer_ID;
```

우의 두 실례는 꼭 같은 결과모임을 돌려준다.

Family_Name	Personal_Name	Order_Number
김	성철	5
김	은희	2
김	철	3
강	권혁	4
최	영실	7
리	성민	6

실례로 Ordered_Items표는 주문번호와 Inventory표의 상품들사이에 련결을 제공한다. 주문번호 2에 대응하는 상품들의 구체적인 목록을 얻기 위하여 다음과 같은 JOIN지령을 작성할수 있다.

```
SELECT Orders.Order_Number, Ordered_Items.Item_Number,
Ordered_Items.Qty, Inventory.Name, Inventory.Description
FROM Orders, Ordered_Items, Inventory
WHERE Orders.Order_Number = Ordered_Items.Order_Number AND
      Inventory.Item_Number = Ordered_Items.Item_Number AND
      Orders.Order_Number = 2;
```

WHERE문절비교에서 리용된 렬들이 다양한 표들의 열쇠렬들이라는데 주의하기 바란다. 결과는 다음과 같다.

Order_number	Item_number	Qty	Name	Description
2	1001	2	고려인삼술	주류
2	1004	1	인풍술	주류
2	1005	4	대동강맥주	청량음료
2	1010	6	박하사탕	당과류

안같기결합

갈기결합외에 매우 드문 일이기는 하지만 안같기(<>)관계에 의한 결합 즉 안같기결합을 리용할수도 있다. 레를 들어 Orders표에 주문번호가 2가 아닌 나머지주문들에 대한 자료를 얻기 위하여 다음과 같이 안같기결합을 리용할수도 있다.

```
SELECT c.Family_Name + c.Personal_Name AS Customers, oi.Qty,
   i.Name, i.Description, i.Cost * 1.6 AS Price_Each,
   i.Cost * 1.6 * oi.Qty AS Price
FROM Orders o, Customers c, Ordered_Items oi, Inventory i
WHERE o.Order_Number = oi.Order_Number AND
   c.Customer_ID = o.Customer_ID AND
   i.Item_Number = oi.Item_Number AND o.Order_Number <> 2;
```

내부결합과 외부결합

우리가 지금까지 학습한 결합은 다 내부결합이다.

내부결합(Inner Join)은 두 표들사이에 존재하며 두 표에서 일치하는 마당값들을 가진 행들만 포함한다. 주문자표와 주문표에서 주문자ID렬이 중첩되여있는 그림 2-1을 보면 내부결합과 외부결합의 의미를 쉽게 리해할수 있다.

내부결합을 리용하면 실레에서 보여준것처럼 주문을 신청한 주문자들만을 렬거할수 있다. 만일 이전에 주문했던적이 있는 임의의 주문날자를 가진 주문자들도 포함한 모든 주문자들의 목록이 요구되는 경우 내부결합으로는 그것을 얻을수 없다.

Family_ Name	Personal_ Name	Sex	Customer_ID	Order_ Number	Order_ Date	Ship_ Date
김	성철	남	100			
김	은희	녀	101	2	2007-04-09	2007-04-11
김	철	남	102	3	2007-04-09	2007-04-11
박	성호	남	106	5	2007-04-10	2007-04-12
강	권혁	남	107			
하	성진	남	108			

그림 2-1. 주문자ID에 의하여 결합된 표

외부결합(Outer Join)은 모임이나 표들의 사귐령역에 속한 레코드들만이 아니라 그 밖에 있는 레코드들도 포함한다. 외부결합조건에는 3가지 류형이 있다.

- LEFT OUTER JOIN(*=)
- RIGHT OUTER JOIN(=*)
- FULL OUTER JOIN

왼쪽외부결합(LEFT OUTER JOIN)연산자들은 결합지령의 왼쪽에 있는 표의 모든 행들을 포함한다. 이것은 아래에 보여준것처럼 어떤 주문도 하지 않은 모든 주문자들을 다 포함한다.

```
SELECT c.Family_Name, c.Personal_Name, o.Order_Date
FROM Customers c LEFT OUTER JOIN Orders o
ON c.Customer_ID = o.Customer_ID;
```

이 질문의 결과모임을 표 2-13에 보여주었다. 주문날자에 기재한 NULL값은 주문자가 주문하지 않았다는것을 의미한다.

표 2-13. 왼쪽외부결합의 결과

Family_Name	Personal_Name	Order_Date
김	성철	2007-04-10
김	은희	2007-04-05
김	철	2007-04-09

Family_Name	Personal_Name	Order_Date
박	성호	NULL
강	권혁	2007-04-09
하	성진	NULL
최	영실	2007-04-14
리	성민	2007-04-12
오	성철	NULL

《왼쪽》과 《오른쪽》은 순전히 SQL문에서 표들의 순서에 의존하기때문에 JOIN 지령에서 표들의 순서를 바꾸면 왼쪽외부결합을 **오른쪽외부결합**(RIGHT OUTER JOIN)으로 바꿀수 있다.

```
SELECT c.Family_Name + c.Personal_Name AS Customers, o.Order_Date
FROM Orders o RIGHT OUTER JOIN Customers c
   ON c.Customer_ID = o.Customer_ID;
```

외부결합지령들은 내부결합에서 리용하였던 속기형식과 류사하게 쓸수도 있다. 왼쪽내부결합을 위한 속기형식은 아래에 보여준것처럼 "*="연산자를 리용한다.

```
SELECT c.Family_Name + c.Personal_Name AS Customers, o.Order_Date
FROM Customers c, Orders o
   WHERE c.Customer_ID *= o.Customer_ID;
```

오른쪽내부결합을 위한 형식은 "=*"연산자를 리용한다.

```
SELECT c.Family_Name + c.Personal_Name AS Customers, o.Order_Date
FROM Orders o, Customers c
   WHERE o.Customer_ID =* c.Customer_ID;
```

외부결합의 속기형식에서 JOIN의 류형은 FROM문절에서 표들의 순서와 "*="연산자에서 별표의 위치에 다같이 의존한다.

완전외부결합(FULL OUTER JOIN)은 두 표로부터 일치하지 않는 모든 행들도 다 포함한다. 례를 들어 Customers표의 어느 항목과도 일치하지 않는 주문자ID를 가진 임의의 주문자를 Orders표에서 찾기 위해서는 두 표로부터 모든 항목들을 보여주는 완전외부결합을 실행한다.

```
SELECT c. Family_Name, c.Personal_Name, o.Order_Date
FROM Customers c FULL OUTER JOIN Orders o
ON c.Customer_ID = o.Customer_ID;
```

이 결합에 의해 생성된 결과모임은 주문자표와 주문표에 있는 모든 항목들을 포함하는 자료들이다. 만일 어떤 리유로 Orders표에 기입된 주문에 대한 주문자가 Customers표에 없다면 아래에서 보는것처럼 표의 아래에 추가적인 행이 생긴다.

표 2-14. 완전외부결합의 결과

Family_Name	Personal_Name	Order_Date
김	성철	NULL
김	은희	2007-04-05
김	철	2007-04-09
박	성호	2007-04-10
강	권혁	NULL
하	성진	NULL
NULL	NULL	2007-04-14

2) NOT EXISTS의 리용

지금까지 두 표로부터 일치하는 마당들을 가진 레코드들을 찾기 위하여 내부결합을 리용하는 방법과 일치하지 않는 마당들도 포함한 모든 레코드들을 찾기 위하여 외부결합을 리용하는 방법을 고찰하였다. 이번에는 다른 표에 대응하는 레코드들을 가지지 않는 어떤 표로부터 레코드들을 찾으려는 경우를 고찰해보자.

주문자표와 주문표를 다시 리용하여 주문을 하지 않은 모든 주문자들을 찾아보자. 방도는 주문표에 존재하지 않는 주문자ID를 가진 주문자레코드를 찾아내는것이다. 이것은 NOT EXISTS를 리용하여 수행한다.

```
SELECT c.Family_Name + c.Personal_Name As Customers
FROM Customers c
WHERE NOT EXISTS
   (SELECT *
   FROM Orders o
   WHERE o.Customer_ID = c.Customer_ID);
```

3) 자체결합

자체결합(self-join)은 하나의 표내에서 자체로 결합을 하는 보통의 SQL결합이다. 표에서 행들이 같은 표의 다른 행들에 대한 참조를 포함할 때 자체결합을 리용한다. 그 실례로 매개 레코드가 Employee_ID에 의해 종업원의 책임자에 대한 참조를 포함하는 종업원표를 들수 있다. 그 책임자도 역시 종업원이므로 그에 대한 정보도 종업원표에 저장된다. 따라서 어떤 종업원의 책임자에 대한 자료를 찾기 위해서는 자체결합을 리용한다.(표 2-15)

표 2-15. 종업원표(Employees)

EMPLOYEE_ID	FAMILY_NAME	PERSONAL_NAME	SUPERVISOR
100	김	성철	104
101	김	은희	107
102	박	철남	107
103	리	영남	105
104	려	수길	99
105	최	창혁	99
106	백	영미	108
107	강	권일	99
108	오	성민	99

결합은 결합해야 할 표들을 식별하는 두개의 표이름을 요구한다. 때문에 표에 대한 참조마다에 별개의 이름을 주기 위하여 표이름 별명을 리용한 자체결합을 만들수 있다.

```
SELECT e.Family_Name, e.Personal_Name,
       boss.Family_Name + boss.Personal_Name AS Boss
FROM Employees e, Employees boss
WHERE e.supervisor = boss.employee_id;
```

앞에서 본 SQL코드는 같은 표에 대하여 E(종업원)와 Boss(책임자)라는 두개의 참조를 만들고 내부결합을 리용하여 그것들을 결합하고있다. 이 방법은 그 표에 대한 하나의 참조로부터 종업원정보를 얻고 다른 참조로부터 책임자정보를 얻을수 있게 한다. 결과표를 아래에 보여주었다.

Family_Name	Personal_Name	Boss
김	성철	려수길
리	영남	최창혁
김	은희	강권일
박	철남	강권일
백	영미	오성민

이것은 다음과 같이 외부자체결합으로 쉽게 바꿀수 있다.

```
SELECT e.Family_Name, e.Personal_Name,
       boss.Family_Name + boss.Personal_Name AS Boss
FROM Employees e, Employees boss
WHERE e.Supervisor *= boss.Employee_id;
```

이 질문은 려수길의 책임자의 Enployee_ID가 Employees표에 없기때문에 그의 책임자는 아래에 보여준것처럼 <NULL>로 나타난다.

Family_Name	Personal_Name	Boss
김	성철	려수길
김	은희	강권일
박	철남	강권일
리	영남	최창혁
려	수길	NULL
최	창혁	NULL
백	영미	오성민
강	권일	NULL
오	성민	NULL

4) 질문결합을 위한 UNION연산자

두개의 분리된 자료원천으로부터 자료를 결합하기 위한 또 하나의 방도는 UNION연산자이다. UNION연산자는 둘 혹은 그 이상의 질문결과들을 단일한 질문결과로 통합하고 임의의 중복행들을 없앤다. UNION과 함께 ALL이 리용될 때에는 중복되는 행까지 다 보여준다.

다음의 실례에서 첫 질문은 《김》가 성을 가진 모든 주문자들의 이름과 주소를 돌

려주며 두번째 질문은 평양시에 있는 모든 주문자들을 돌려준다. UNION연산자는 평양시의 《김》가성을 가진 주문자들의 중복레코드들을 제거하고 결과들을 통합한다.

```
SELECT Family_Name, Personal_Name, State, City, District, Dong
FROM Customers
WHERE Family_Name = '김'
UNION
SELECT Family_Name, Personal_Name, State, City, District, Dong
FROM Customers
WHERE State = '평양시'
ORDER BY Family_Name, Personal_Name;
```

이 질문의 결과는 다음표와 같다. 마지막 질문뒤에 ORDER BY문절을 추가하여 결합된 결과모임을 정렬한다.

Family_ Name	Personal_ Name	State	City	District	Dong
강	권혁	평양시	평양	중구역	류성동
김	성철	평양시	평양	중구역	교구동
김	은희	평양시	평양	보통강구역	대보동
김	철	함경남도	함흥	<NULL>	사포동
오	성민	평양시	평양	평천구역	안산2동

매 질문에서 꼭 같은 렬들을 사용할 필요는 없지만 렬의 총수와 렬의 자료형들은 일치해야 한다. 렬의 수가 다른 두 결과모임을 결합하는 경우에는 렬번호를 리용한 ORDER BY문절을 적용하여야 한다.

❖ 데카르트적에 대한 리해 ❖

결합의 데카르트적은 한 표의 모든 레코드가 다른 표의 모든 레코드와 결합될 때 발생한다. 그러므로 100행짜리 두개의 표에 대한 데카르트적은 10000행으로 된다. 데카르트적은 보통 WHERE문절이 적합치 않거나 존재하지 않을 때 발생하는 오유이다. 실례에서와 같은 작은 표인 경우에는 이것이 그리 큰 문제가 아니지만 큰 자료기지 례컨대 1000개의 행에 대한 데카르트적을 생성하는데 걸리는 시간은 상당히 길어질수 있다.

EXCEPT연산자

EXCEPT연산자는 첫 질문이 돌려주는 결과모임에서 두번째 질문이 돌려주는 결과모임을 제외한 행들의 모임을 돌려준다. 다음의 문은 평안남도를 제외한 도들에서 사는 주문자들 가운데서 《김》가성을 가진 모든 주문자들의 이름과 주소를 돌려준다.

```
SELECT Family_Name, Personal_Name, State, City, District, dong
FROM Customers
WHERE Family_Name = '김'
EXCEPT
SELECT Family_Name, Personal_Name, State, City, District, dong
FROM Customers
WHERE State = '평안남도'
```

INTERSECT연산자

INTERSECT연산자는 두 질문에 존재하는 행들중 중복행들을 제거한 결과모임을 작성한다. INTERSECT와 함께 ALL을 리용할 때에는 중복행들이 제거되지 않는다. 다음의 문은 평양시에 살고있는 《김》가성을 가진 주문자들에 대한 이름과 주소를 돌려준다.

```
SELECT Family_Name, Personal_Name, State, City, District, Dong
FROM Customers
WHERE Family_Name = '김'
INTERSECT
SELECT Family_Name, Personal_Name, State, City, District, Dong
FROM Customers
WHERE State = '평양시'
```

제5절. 자료조종언어

자료조종언어는 자료기지를 관리하며 사용자접근특권과 같은것들을 조종하기 위한 도구들을 제공한다. 사용자들을 관리하는것은 자료기지관리의 중요한 측면의 하나이다.

사용자는 자료기지에 접근하는 임의의 사람이다. 사용자접근특권에는 자료기지의 제한된 부분에 대한 읽기만이 가능한 낮은 준위로부터 전체 관계형자료기지관리체계에 대한 무제한한 접근에 이르기까지 각이한 특권이 있다.

2.5.1. 사용자관리

자료기지관리자가 자료기지에 개별적인 사용자들을 추가하려면 CREATE USER지령을 리용하여 자료기지사용자들을 창조하여야 한다. 사용자를 창조할 때 통과암호, 정해진 기초허가권과 만기날자를 모두 한번의 지령으로 할당할수 있다. 또한 그 사용자를 이미 있는 사용자그룹에 추가할수도 있다.

사용자를 창조한 후 사용자의 특권준위를 변경시킬 필요가 제기될수 있다. 이때에는 ALTER USER지령을 리용한다.

어떤 사용자에 대하여 자료기지에 대한 접근을 완전히 차단하기 위해서는 DROP USER지령을 리용한다.

사용자특권

관계형자료기지관리체계는 사용자들에게 할당할수 있는 **특권**(privilege)들의 모임을 정의하는데 이 특권들은 사용자가 자료기지의 객체들에 실시할수 있는 작용들과 대응된다.

사용자특권은 두가지 서로 다른 준위에서 할당할수 있다. 사용자들은 READ, MODIFY, WRITE와 같이 그들이 수행할수 있는 작용의 형태에 대한 준위와 그들이 접근할수 있는 자료기지객체의 류형에 대한 준위의 두 측면에서 특권을 가진다.

접근준위특권(access-level privilege)에는 일반적으로 다음과 같은것들이 있다.

- 주어진 봉사기상에 있는 모든 자료기지들에 접근할수 있는 공개준위
- 주어진 자료기지에 있는 모든 표들에 접근할수 있는 자료기지준위
- 주어진 표의 모든 렬들에 접근할수 있는 표준위
- 주어진 표에서 몇가지 렬들에만 접근할수 있는 렬준위

표준적으로 사용자특권에 대한 관리는 자료기지관리자에 의하여 조종된다.

사용자의 역할을 정의하면 사용자특권들이 할당된다. 사용자그룹과 역할도 역시 사용자와 같이 SQL지령으로 처리된다.

대부분의 관계형자료기지관리체계에서는 다음과 같은 역할들을 지원해준다.

- Owner - 자료를 읽고 쓸수 있으며 자료기지와 그 구성요소들을 작성, 변경 및 삭제할수 있는 사용자
- Writer - 자료의 읽기와 함께 쓰기가 허가된 사용자
- Reader - 자료기지에서 자료를 읽어보기만 하고 써넣을수는 없는 사람
- Public - 특권준위의 개념에서 특권준위가 가장 낮은 사람

사용자역할들은 자료기지관리자가 시간을 절약하도록 하기 위한 순수 관리적인 기능이다. 역할들은 필요에 따라 자료기지관리자가 정의할수 있다.

사용자그룹관리

많은 체계들에서는 자료기지관리자가 사용자들을 같은 특권준위를 가진 론리적인 그룹으로 가를수 있게 한다. 그룹들은 개별적인 사용자들과 거의 같은 방식으로 작성된다. 그룹창조를 위한 일반적인 문장구성법은 다음과 같다.

```
CREATE GROUP 그룹이름 WITH USER 사용자1, 사용자2
```

사용자제거와 마찬가지로 사용자그룹도 DROP지령으로 제거할수 있다.

```
DROP GROUP 그룹이름
```

어떤 그룹에 사용자들을 추가하려면 ALTER GROUP ... ADD지령을, 제거하려면 ALTER GROUP ... DROP지령을 리용한다.

```
ALTER GROUP 그룹이름 ADD USER 사용자이름 [, ... ]
ALTER GROUP 그룹이름 DROP USER 사용자이름 [, ... ]
```

2.5.2. 사용자특권의 허가와 취소

GRANT지령은 자료기지에서 다양한 조작을 수행하는데 필요한 접근특권을 사용자들에게 할당하는데 리용된다. GRANT지령은 이밖에도 그 사용자가 다른 사용자들에게 어떤 특권을 허용하도록 하는데도 리용할수 있다. 또한 사용자에게 모든 부분표들과 련관된 표들에 대한 특권들을 허가하게 하는 옵션도 있다. 이 두가지 형태의 GRANT지령을 아래

에 보여주었다.

```
GRANT 특권 ON 표이름 TO 사용자;
GRANT SELECT ON PRODUCTS WITH GRANT OPTION TO 사용자;
```

REVOKE지령은 어떤 사용자에게 허가된 특권을 취소하는데 리용된다. GRANT지령과 같이 이 지령은 다양한 준위들에 적용될수 있다. 이 지령의 정확한 문장구성법은 자료기지마다 다를수 있다. 례를 들어 다음의 지령은 사용자에게 할당되여있는 제품표에 대한 SELECT특권을 취소한다.

```
REVOKE SELECT ON PRODUCTS FROM 사용자;
```

제6절. 저장수속의 작성과 리용

2.6.1. 저장수속의 개념

저장수속(stored procedure)은 사용자로부터 파라메터들을 받고 또 사용자에게 결과값들을 돌려줄수 있는 미리 저장된 SQL문들의 모임이다. 즉 저장수속은 SQL로 작성된 메쏘드나 함수라고 생각할수 있다.

저장수속은 다음의 사항들을 포함한 몇가지 우점을 가진다.

- 저장수속은 미리 콤파일되여있기 때문에 빨리 실행된다.
- 저장수속은 공통적인 과제들을 수행하기 위한 표준방식을 제공한다.

거의 모든 SQL문이 저장수속으로 리용될수 있다. 저장수속을 만들려면 저장수속이름과 변수들을 렬거해야 한다.

```
CREATE PROCEDURE 저장수속이름
   @parameter1 자료형,
   @parameter2 자료형 = 기정값,
   @parameter3 자료형 OUTPUT
AS
   SQL지령들 [...]
```

변수이름들은 첫 문자를 @기호로 시작한다. 그 외에 식별자를 위한 일반규칙에 맞아야 한다. 변수이름들은 표이름, 렬이름, 기타 자료기지객체들의 이름을 대신 리용할수 없다. 변수이름들은 저장수속에 값들을 넘기고 그로부터 값들을 받는데만 리용될수 있다.

저장수속에서는 변수이름과 함께 그 변수의 자료형을 지적해야 한다. 저장수속에서 리용할수 있는 파라메터들은 임의의 자료형들이 다 될수 있다. 또한 실례에 보여준것처럼 변수에 대한 기정값을 지적할수도 있다.

저장수속이 호출자에게 돌려주는 값이 있는 경우 돌려주는값에 사용할 변수와 자료형 다음에 반드시 OUTPUT열쇠단어가 있어야 한다. AS는 저장수속의 본체를 이루는 SQL문의 시작을 알리는데 리용된다. 파라메터가 없는 가장 단순한 저장수속의 실례를 아래에 보여준다.

```
CREATE PROCEDURE LIST_ORDERS_BY_STATE
AS
SELECT
   o.Order_Number,
   c.Family_Name + c.Personal_Name AS Name,
   c.State
FROM Customers c, Orders o
   WHERE c.Customer_ID = o.Customer_ID
   ORDER BY c.State, c.Family_Name;
```

저장수속의 호출은 간단히 저장수속의 이름을 지적하면 된다.

```
LIST_ORDERS_BY_STATE;
```

저장수속은 다음의 결과를 돌려준다.

Order_Number	Name	State
3	김 철	함경남도
7	최영실	강원도
6	리성민	평안북도
4	강권혁	평양시
5	김성철	평양시
2	김은희	평양시

2.6.2. 저장수속에서 입력파라메터의 리용

다음의 코드는 저장수속에서 입력파라메터를 리용하는 방법을 보여준다.

이 저장수속은 HTML폼에서 입력된 내용을 다루기 위하여 설계된것이다. 변수이름들이 렬이름들과 다르다는데 대하여 주의하기 바란다.

```
CREATE PROCEDURE INSERT_CUSTOMERS
@FName VARCHAR(20), @PName VARCHAR(20), @SX VARCHAR(2),
@ST VARCHAR(8), @CT VARCHAR(8), @DT VARCHAR(8), @DO VARCHAR(8), @NB
VARCHAR(10)
AS
INSERT INTO CUSTOMERS
   (Family_Name, Personal_Name, Sex,
    State, City, District,Dong, Neighbor)
VALUES
   (@FName, @PName, @SX,
    @ST, @CT, @DT, @DO, @NB);
```

이 저장수속을 호출하는 SQL문은 앞에서와 매우 류사하다. 다만 차이는 호출시 HTML폼으로부터 얻은 입력파라메터를 리용한것이다.

```
INSERT_CUSTOMERS '리', '영덕', '남', '강원도', '원산시', NULL, '석금동', '26'
```

2.6.3. 저장수속에서 출력파라메터의 리용

출력파라메터를 리용하는 저장수속을 만드는것도 역시 간단하다. 실례로 사용자가 입력한 이름과 통과암호가 정확한가 하는 확인통보문을 돌려주는 저장수속을 보기로 하자.

```
CREATE PROCEDURE CHECK_USER_NAME
   @FName varchar(30),
   @PName varchar(20),
   @PassFail varchar(20) OUTPUT
AS
   IF EXISTS(SELECT * FROM Customers
             WHERE Family_Name = @FName
             AND
             Personal_Name = @PName)
      BEGIN
         SELECT @PassFail = "PASS"
      END
```

```
    ELSE
        BEGIN
            SELECT @PassFail = "FAIL"
        END
```

@PFValue이라는 변수를 선언하고 그것을 아래와 같이 출력파라메터로 저장수속에 넘겨주면 저장수속에서의 출력결과가 @PFValue변수에 들어오게 된다. 이 실례에서 결과는 새로운 PWCHECK표에 저장된다.

```
DECLARE @PFValue VARCHAR(20)
EXECUTE CHECK_USER_NAME '김', '성철', @PFValue OUTPUT
INSERT INTO PWCHECK
VALUES ('김', '성철', @PFValue)
```

2장에 대한 요약

이 장에서는 SQL에 대하여 간단하면서도 알기 쉽게 개괄하였다. 이 장을 읽은 다음 독자들은 자료기지를 작성하고 어느정도 복잡한 질문들을 수행할수 있는 SQL문을 작성할수 있어야 한다.

특히 다음의 경우들에 대처한 SQL의 리용을 습득해야 한다.

- 자료기지와 표의 작성
- 자료기지질문
- 표들을 결합하기 위한 기본열쇠와 외부열쇠의 리용
- 자료기지보안관리

3장에서는 자바자료기지접속(JDBC)에 대하여 학습하게 된다.

제 3 장. JDBC 입문

JDBC는 자바프로그람에서 표형식의 자료원천에 접근할수 있게 하는 자바자료기지접속 응용프로그람대면부(Java Database Connectivity API)이다. JDBC는 넓은 범위의 SQL자료기지에 대한 접속을 제공하는것과 함께 펼친표나 평문파일과 같은 다른 표형식의 자료원천들에도 접근할수 있게 한다. 독자들은 보통 JDBC를 Java Database Connectivity의 준말로 생각할수 있는데 등록된 응용프로그람대면부의 이름이 실제로 JDBC이다.

이 장에서는 JDBC에 대한 개념과 JDBC를 리용하여 자바응용프로그람을 작성하는데 필요한 기초지식들을 취급한다.

제1절. JDBC의 개념

JDBC는 임의의 특정한 SQL실현과는 독립으로 기본적인 SQL기능들을 지원하도록 설계된 저준위 API이다. 이것은 JDBC설계의 초점이 SQL문들을 그대로 실행하고 그 결과들을 검색하는데 있다는것을 의미한다. JDBC는 SQL자료기지들에 접근하기 위한 프로그람작성의 국제표준이며 마이크로쏘프트의 공개자료기지접속(Open Database Connectivity: ODBC)대면부의 기초인 X/Open SQL Call Level Interface에 기초하고있다.

JDBC 2.0 API는 두개의 패키지 즉 JDBC 2.0 핵심부 API로 알려진 `java.sql`과 JDBC표준확장인 `javax.sql`을 포함하고있다. 이것들은 자바를 리용하여 자료기지응용프로그람을 개발하기 위한 필수적인 클라스들을 포함하고있다.

2001년 10월에 발표된 JDBC 3.0은 다양한 자료형의 지원, 추가적인 MetaData능력들, 많은 대면부들에 대한 확장을 포함하는 여러가지 특징들을 가진다.

JDBC확장패키지(`javax.sql`)는 선택적패키지들인 JNDI(Java Naming and Directory Interface), JTS(Java Transaction Service)와 같은 자바가동기반의 다른 토막들과 밀접하게 련관된 JDBC API의 주요부분들을 포함한다. 더우기 접속공유와 RowSet와 같은 JDBC API핵심부로부터 쉽게 분리할수 있는 일부 고급한 특징들이 이 확장패키지에 추가되였다.

JDBC의 주되는 장점은 그것이 임의의 관계형자료기지들과 꼭 같은 방식으로 동작한다는것이다. 다시말하여 Oracle, Sbase, SQL Server 등 각이한 제품의 관계형자료기지

에 접근하는데 서로 다른 프로그람을 작성할 필요가 없다. JDBC는 각이한 자료기지-접속모듈들의 다양성우에 유일한 SQL대면부를 제공한다.

3.1.1. JDBC의 주요기능

JDBC의 기능은 크게 3가지로 나눌수 있다.

- 자료기지 혹은 기타 표형식의 자료원천들에 대한 접속확립
- 자료기지에 SQL지령넘기기
- 결과처리

목록 3-1은 다양한 상품들의 품명(Name), 품종(Description), 수량(Qty), 원가(Cost)를 포함하고있는 Stock(구입품)표가 들어있는 상품목록자료기지 Inventory에 접근하는 실례코드이다. 코드에는 JDBC를 리용하여 자료원천에 접근하기 위한 3가지 단계가 명백히 서술되여있다.

목록 3-1. JDBC기능의 간단한 실례

```
package JavaDB_Bible.ch03.sec01;

// JDBC핵심패키지들을 반입
import java.sql.*;

public class JdbcDemo{
  public static void main(String args[]) {
    int qty;
    float cost;
    String name;
    String desc;

    // SQL질문문자렬
    String query = "SELECT Name, Description, Qty, Cost FROM Stock";

    try {
      // JDBC구동프로그람 적재
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      // 접속얻기
      Connection con = DriverManager.getConnection("jdbc:odbc:Inventory");
      Statement stmt = con.createStatement();
```

```
      // 질문실행
      ResultSet rs = stmt.executeQuery(query);

      // 결과분석
      while (rs.next()) {
        name = rs.getString("Name");
        desc = rs.getString("Description");
        qty = rs.getInt("Qty");
        cost = rs.getFloat("Cost");
        System.out.println(name + "\t" + desc + "\t" + qty + "\t" + cost + "원");
      }
      con.close();
    }
    catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
  }
}
```

우의 실례에서는 JDBC API를 리용하여 자료기지에 접근하고 ResultSet로부터 자료를 검색하는데 필요한 다음의 기본적인 단계들을 보여주고있다.

- JDBC구동프로그람을 적재한다.
- 자료기지에 대한 접속을 얻는다.
- SQL문을 생성한다.
- SQL문을 실행한다.
- ResultSet로부터 자료를 추출한다.

ResultSet는 결과들을 완전히 장악하는데 필요한 메쏘드들을 제공하며 각각의 류형에 적합한 메쏘드들을 리용하여 개별적인 자료기지마당들을 얻는다. 코드의 실행결과는 다음과 같다.

Name	Description	Qty	Cost
랭천사이다	청량음료	23	2.5원
평양맥주	청량음료	43	3.4원
모란과자	당과류	20	1.5원

JDBC API는 BLOB, CLOB, ARRAY, REF, STRUCTURE와 같은 SQL99의 고급자료형에 대한 지원을 비롯하여 SQL자료형과 Java/JDBC자료형들사이의 표준대응을 정의한다.

3.1.2. JDBC일치

java.sql.Driver의 jdbcCompliant()메쏘드는 구동프로그람이 JDBC에 일치되는가를 검사한다. 이 메쏘드는 구동프로그람이 JDBC일치에 통과되면《참》, 통과되지 못하면《거짓》을 돌려준다. Sun회사에서는 다음 3가지 준위의 JDBC일치성을 정의하고있다.

1. JDBC 1.0 API일치는 다음과 같은 대면부의 실현을 요구한다.
   - java.sql.Driver
   - java.sql.DatabaseMetaData(JDBC 2.0과 3.0확장에서 정의된 부분들은 제외)
   - java.sql.ResultSetMetaData(JDBC 2.0과 3.0확장에서 정의된 부분들은 제외)
   - java.sql.Connection
   - java.sql.Statement
   - java.sql.CallableStatment
   - java.sql.PreparedStatement
   - java.sql.ResultSet

2. JDBC 2.0 API일치는 다음과 같은것을 요구한다.
   - JDBC 1.0 API일치
   - JDBC 2.0에서 정의된 DatabaseMetaData대면부확장의 완전한 실현
   - 추가적인 JDBC 2.0 ResultSet메쏘드들의 실현

3. JDBC 3.0 API 일치는 다음과 같은것을 요구한다.
   - JDBC 2.0 API일치
   - java.sql.ParameterMetaData의 실현
   - java.sql.SavePoint의 실현
   - JDBC 3.0에서 정의된 DatabaseMetaData대면부확장의 완전한 실현

구동프로그람개발자들은 JDBC API와 함께 쓸수 있는 검사도구들을 리용하여 자기들이 만든 구동프로그람이 JDBC일치규격에 만족되는가를 확인할수 있다.

제2절. JDBC의 기본동작

JDBC핵심 API에서 기본적인 대면부들은 다음과 같다.

- java.sql.DriverManager－DriverManager는 JDBC구동프로그람들을 적재하는외에 적합한 구동프로그람에 대한 접속을 돌려주는 역할을 한다. getConnection()메쏘드가 호출되면 DriverManager는 이미 등록된 구동프로그람들을 조사하여 호출에서 제공된 URL에 맞는 적당한 구동프로그람을 찾는다.
- java.sql.Driver － Driver객체는 DriverManager가 넘겨준 URL에로의 접속가능성을 확인하는 acceptsURL(String url)메쏘드를 수행한다.
- java.sql.Connection － Connection객체는 JDBC API와 URL에 지정된 자료기지관리체계사이의 접속을 제공한다. Connection은 어떤 특정한 자료기지와의 대화접속을 나타낸다.
- java.sql.Statment － Statement객체는 주어진 접속우에서 SQL문을 실행하기 위한 포함기(container)로 작용한다.
- java.sql.ResultSet－ResultSet객체는 주어진 Statement의 결과들에 대한 접근을 조종한다. ResultSet객체는 유표를 통하여 앞뒤로 이동할수 있고 취득자메쏘드를 리용하여 자료에 접근할수도 있다.

3.2.1. DriverManager

java.sql.DriverManager는 JDBC구동프로그람들을 관리하기 위한 기본적인 봉사를 제공한다. DriverManager는 초기화기간에 jdbc.drivers체계속성에서 참조되는 구동프로그람클라스들을 적재한다. 한편 응용프로그람은 Class.forName()을 통해서도 임의의 시간에 JDBC구동프로그람을 적재할수 있다. 이것은 사용자들이 자기의 응용프로그람이 리용할 JDBC구동프로그람들을 주문할수 있게 한다.

새롭게 적재된 구동프로그람클라스들은 registerDriver()메쏘드를 호출하여 DriverManager에 자기를 등록한다. 보통 구동프로그람은 이 메쏘드에 대한 호출을 내적으로 진행한다.

getConnection()메쏘드가 호출될 때 DriverManager는 초기화시에 적재된 구동프로그람들과 현재 애플레트 혹은 프로그람에서 명시적으로 적재된 구동프로그람들속에서 적합한 구동프로그람을 찾는다. 이때 DriverManager는 등록된 모든 구동프로그람들을 조사하여 그 구동프로그람들의 acceptsURL()메쏘드에 자료기지에 대한 URL을 넘기는 방법으로 수행된다.

사용자는 자료기지의 URL외에 추가적인 인수들을 넘겨줄수 있다.

getConnection()메쏘드의 형식에는 3가지가 있다.

```
public static synchronized Connection getConnection(String url)
   throws SQLException

public static synchronized Connection getConnection(String url,
   String user, String password) throws SQLException

public static synchronized Connection getConnection(String url,
   Properties info) throws SQLException
```

주의: 구동프로그람을 탐색할 때 JDBC는 주어진 URL에 성공적으로 접속할수 있는 처음으로 발견된 구동프로그람을 리용한다. JDBC는 sql.drivers목록에 지적된 구동프로그람들부터 주어진 순서대로 시작한다. 다음에는 구동프로그람들이 적재된 순서로 조사한다.

3.2.2. JDBC구동프로그람

개별적인 자료기지들과 접속하는데서 JDBC는 매 자료기지에 적합한 구동프로그람을 요구한다. JDBC구동프로그람들의 기본적인 류형은 4가지이다. 첫번째와 두번째 류형은 응용프로그람을 작성하는 프로그람작성자들을 위한것이고 세번째와 네번째 류형은 주로 미들웨어나 자료기지제작업자들을 위한것이다.

4가지 류형의 JDBC구동프로그람들은 다음과 같다.

- 류형 1: JDBC-ODBC bridge plus ODBC driver
- 류형 2: Native-API partly Java driver
- 류형 3: JDBC-Net pure Java driver
- 류형 4: Native-protocol pure Java driver

JDBC-ODBC bridge는 ODBC구동프로그람들을 통하여 JDBC에 대한 접근을 제공한다. ODBC는 자바환경이 아닌곳에서 자료기지들에 접속하기 위하여 널리 쓰인다. ODBC는 관계형자료기지들에 대한 접근에서 가장 널리 리용되고있는 프로그람작성대면부이다.

JDBC-ODBC bridge의 우점은 다음과 같다.

- 거의 모든 가동기반상에 있는 거의 모든 자료기지들에 접속할수 있다.
- 일부 낮은 급의 탁상형자료기지들과 응용프로그람들에 접근하기 위한 유일한 방도일수 있다.

JDBC-ODBC bridge의 기본결함은 다음과 같이 찾아볼수 있다.

- ODBC구동프로그람들이 대상기계에 적재되여있어야 한다.

- JDBC와 ODBC사이의 변환이 프로그람의 성능에 영향을 미친다.

류형 2의 구동프로그람들은 자료기지체계와 통신하는데 본연의 API를 리용한다. 자료기지조작을 수행하는 API기능들을 호출하는데 자바본연의 메쏘드들이 리용된다.

류형 2의 큰 우점은 류형 1보다 속도가 빠른것이다.

류형 2의 기본결함은 다음과 같다.

- 류형2의 구동프로그람들은 대상기계 본연의 코드를 요구한다.
- 이 구동프로그람들이 의존하는 Java Native Interface가 Java가상기계의 제작자들에 따라 똑같이 실현되지 않는다.

류형 3의 구동프로그람들은 JDBC호출을 자료기지관리체계독립의 망규약으로 변환하며 망규약은 봉사기에 의해 자료기지관리체계규약으로 변환된다.

이 구동프로그람들의 우점은 다음과 같다.

- 의뢰기측에 본연의 2진코드가 있을것을 요구하지 않는다.
- 의뢰기설치를 필요로 하지 않는다.
- HTTP턴넬화와 같은 여러가지 망관련 추가선택항목들을 지원한다.

류형 3 구동프로그람들의 결함은 그 구성방식이 망대면부에 따라 복잡하므로 설치하기가 힘들수 있다는것이다.

류형 4의 구동프로그람은 100% 자바구동프로그람 본연의 규약이다. 이것은 자바의뢰기에서 자료기지관리체계봉사기를 직접 호출할수 있게 한다. 류형 4의 구동프로그람은 100% 자바로 작성되였기때문에 응용프로그람이 구동프로그람을 어디서 찾아야 하는가를 지시하는 외에 의뢰기에 다른 구성을 요구하는것이 없다. 이 구동프로그람들은 자료기지제작업체들에 의해 제공되며 류형 1보다 훨씬 속도가 빠르다.

3.2.3. JDBC자료원천

JDBC 2.0 표준확장 API에 도입된 `DataSource`대면부는 오늘날 자료원천에 대한 접속을 만드는데서 `DriverManager`클라스보다 우선적인 선택안이다. 이 자료원천은 관계형자료기지로부터 표형식의 파일에 이르기까지 임의의 형태일수 있다.

`DataSource`객체는 다음의 3가지로 실현될수 있다.

- 분산거래에 리용되지 않는 표준 `Connection`객체들을 낳는 기초 `DataSource`
- 접속공용을 지원하는 `DataSource`
- 둘이상의 자료기지관리체계봉사기에 접근할수 있는 분산거래들을 지원하는 `DataSource`

접속공용(connection pooling)이란 자료기지접근을 위하여 일단 이루어진 접속들을 해방하지 않고 저축해두었다가 다음번의 자료기지접근에 재사용하는것을 말한다. 접속을 공동으로 재사용하면 자료기지에 접근할 때마다 새로운 접속을 만드는것으로 인한 성능저하를 피할수 있다.

분산거래는 여러개의 자료기지봉사기들에 있는 표들을 포함한다. JDBC DataSource는 분산거래를 위한 접속이 가능하도록 실현될수 있다. 이러한 종류의 DataSource은 항상 접속공용이 가능하도록 실현된다.

DataSource객체들은 접속공용과 분산거래를 제공할수 있을뿐만아니라 이동하기 쉽고 보수하기 쉬운 점이 있다. 이 특징들로 하여 DataSource객체들은 자료원천에 대한 접속을 얻는데서 우선적인 수단으로 된다.

자료원천과 자바이름달기 및 등록부대면부

DataSource객체는 생성되면 표준적으로 JNDI이름달기봉사에 등록된다. 응용프로그람은 체계구성과는 독립으로 이름달기봉사로부터 이름에 의해 DataSource객체를 검색할수 있다.

JNDI는 자바응용프로그람들에 대한 이름달기 및 등록부기능을 제공한다. 그것은 임의의 특정한 등록부-봉사실현과 독립으로 정의되였기때문에 다양한 등록부들에 공통적인 방법으로 접근할수 있게 해준다.

JNDI이름달기봉사는 이름에 따라 파일들을 찾고 사용할수 있게 하는 파일등록부와 류사하다. 이 경우 JNDI이름달기봉사는 자료원천이 자기한테 등록될 때 할당된 론리이름을 리용하여 DataSource를 찾는데 리용된다.

어떤 객체와 어떤 이름을 련관시키는것을 **맺기**(binding)라고 한다. 실례로 파일들은 파일이름과 맺기된다. 객체들의 찾기, 맺기, 풀기, 이름변경과 보조콘텍스트들을 생성하고 소멸하는 핵심 JNDI대면부는 Context대면부이다.

Context대면부에는 다음의 메쏘드들이 정의되여있다.

- bind(String name, Object obj) - 객체와 그 객체에 할당된 이름을 맺어준다.
- listBindings(String name) - 지명된 콘텍스트속에 매여있는 이름들을 그 이름들에 매인 객체들과 함께 렬거한다.
- lookup(String name) - 지명된 객체를 검색한다.

자료원천기초실현의 배비와 리용

JDBC의 DataSource객체는 자료원천이름(DSN)과 자료원천이 상주하는 봉사기이름, 포구번호와 같은 속성들의 모임으로서 자료를 찾는데 필요한 정보를 가지고있다.

DataSource객체의 배비는 다음의 3가지 과제로 구성된다.

- DataSource클라스의 구체례생성
- 생성된 구체례의 속성설정
- JNDI이름달기봉사에 등록

첫 단계는 BasicDataSource객체를 생성하고 ServerName, DataBaseName, Description 등의 속성들을 설정하는것이다.

```
com.dbaccess.BasicDataSource ds = new com.dbaccess.BasicDataSource();
ds.setServerName("Jupiter");
ds.setDatabaseName("CUSTOMERS");
ds.setDescription("Customer database");
```

이제는 BasicDataSource객체를 JNDI이름봉사에 등록할수 있는 준비가 되였다. JNDI API는 다음과 같은 방법으로 InitialContext객체를 생성하고 BasicDataSource객체 ds와 론리이름 jdbc/customerDB를 맺기한다.

```
Context ctx = new InitialContext();
ctx.bind("jdbc/customerDB", ds);
```

jdbc라는 앞붙이는 뿌리등록부밑에 보조등록부가 있듯이 초기콘텍스트밑에 있는 JNDI의 보조콘텍스트이다. 보조콘텍스트 jdbc는 DataSource객체들에 매인 론리이름들에 예약되여있으며 항상 자료원천에 대한 론리이름의 첫 부분을 이룬다.

DataSource를 리용하여 접속을 얻기 위해서는 간단히 JNDI의 Context를 참조하고 그의 lookup()메쏘드에 DataSource객체의 이름을 넣어준다. 그러면 lookup()메쏘드가 그 이름에 매여있는 DataSource객체를 돌려준다.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/customerDB");
Connection con = ds.getConnection("myUserName", "myPassword");
```

주의: BasicDataSource객체는 제작업체에 따라 이름이 다를수 있다. 실례로 Opta2000구동프로그람은 BasicDataSource를 TdsDataSource로 호출한다. DataSource의 getConnection메쏘드에서 돌려주는 Connection객체는 DriverManager.getConnection메쏘드가 돌려주는 Connection객체와 같다.

DataSource객체는 사용자가 접속공용이나 분산거래를 포함하는 프로그람을 작성하지 않는한 리용하지 않을수도 있다. 접속공용 혹은 분산거래를 포함하는 응용프로그람들을 작성하는 경우 접속공용 혹은 분산거래능력이 내장된 DataSource객체를 리용하는것이 좋다.

제3절. 접속공용과 분산거래

흔히 자원들의 창조와 파괴는 시간을 많이 요구하며 응용프로그람의 효률을 떨어뜨린다. 자원공용은 어떤 작업을 위해 새로운 자원을 생성하고 그 작업이 끝나는 즉시 그 자원을 파괴하는것으로 인한 부가적인 처리시간을 피할수 있게 한다. 이 절에서는 접속공용과 그것을 기초로 하는 분산거래에 대하여 설명한다.

3.3.1. 공용접속의 창조와 배비

접속공용을 실현하는 객체들은 ConnectionPoolDataSource대면부를 리용하여 창조한다. 이 대면부를 실현하는 접속객체들은 일반적으로 JNDI봉사에 등록된다.

공용(혹은 저축되는)접속을 만들어내는 DataSource객체를 배비하려면 우선 ConnectionPoolDataSource객체를 창조하고 속성들을 적당히 설정해야 한다.

```
ConnectionPoolDataSource cpds = new ConnectionPoolDataSource();
cpds.setServerName("Jupiter");
cpds.setDatabaseName("CUSTOMERS");
cpds.setPortNumber(9001);
cpds.setDescription("Customer database");
```

다음 ConnectionPoolDataSource객체는 JNDI이름달기봉사에 등록된다.

```
Context ctx = new InitialContext();
ctx.bind("jdbc/pool/customerDB", cpds);
```

주의: cpds와 련관된 론리이름은 보조콘텍스트 jdbc아래에 추가된 보조콘텍스트 pool을 가지고있다.

ConnectionPoolDataSource객체가 JNDI이름달기봉사에 등록된 다음 그것과 작업하기 위하여 실현되는 DataSouce객체를 배비하여야 한다.

접속에 필요한 정보는 이미 ConnectionPoolDataSource객체에 설정되였으므로 DataSource객체에서는 다음의 두가지 속성을 설정해야 한다.

- DataSourceName
- Description

DataSourceName은 아래에 보여준것처럼 ConnectionPoolDataSource의 론리이름으로 설정된다.

```
PooledDataSource ds = new PooledDataSource();
ds.setDescription("Customer database pooled connection source");
ds.setDataSourceName("jdbc/pool/customerDB");
Context ctx = new InitialContext();
ctx.bind("jdbc/customerDB", ds);
```

이제는 응용프로그람에서 자료기지에 대한 공용접속들을 얻기 위하여 리용할수 있는 DataSource객체가 배비되였다.

경고: 메쏘드가 례외를 던지는 경우에도 접속이 닫기고 그 접속이 저축에 돌려지도록 마지막 블로크에서 공용접속들을 닫는것이 중요하다.

DataSource객체는 분산거래를 실현하는데도 리용할수 있다.

3.3.2. 분산거래

3층구성방식의 분산거래에서는 하나이상의 자료기지봉사기들에 있는 자료에 접근할 필요가 제기된다. 이런 경우 중간층이 DataSource를 리용하여 분산거래를 위한 접속을 만들어 효률적인 처리를 한다.

접속공용과 관련하여 두개의 클라스가 배비되여야 한다.

- 분산거래를 지원하는 XAConnections를 낳는 XADataSource
- 그것과 작업하도록 실현된 DataSource객체

분산거래를 위한 접속을 낳도록 실현된 DataSource는 거의 항상 공용되는 접속을 낳도록 실현된다.

실제상 XAConnections대면부는 PooledConnection대면부를 확장한것이다.

우선 XADataSource객체가 배비되여야 한다. 그러자면 다음과 같이 XATransactionalDS의 구체례를 창조하고 속성들을 설정한다.

```
XATransactionalDS xads = new XATransactionalDS();
xads.setServerName("Jupiter");
xads.setDatabaseName("CUSTOMERS");
xads.setPortNumber(9001);
xads.setDescription("Customer database");
```

다음 XATransactionalDS는 JNDI이름달기봉사에 등록되여야 한다.

```
Context ctx = new InitialContext();
ctx.bind("jdbc/xa/CustomerDB", xads);
```

마지막으로 DataSource객체가 xads와 호상작용하도록 실현되며 다른 XADataSource객체들이 배비된다.

```
TransactionalDS ds = new TransactionalDS();
ds.setDescription("Customers distributed transaction connections source");
ds.setDataSourceName("jdbc/xa/CustomerDB");
Context ctx = new InitialContext();
ctx.bind("jdbc/CustomerDB", ds);
```

이제는 TransactionalDS와 XATransactionalDS클라스들의 구체례가 배비되였으며 프로그람에서는 DataSource를 리용하여 CUSTOMERS자료기지에 대한 접속을 얻을수 있게 되였다. 그러면 이 접속은 분산거래에 리용될수 있다. 이때 접속을 얻기 위한 코드는 공용접속을 얻기 위한 코드와 대단히 류사하다.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbe/CustomerDB");
Connection con = ds.getConnection("myUserName", "myPassword");
```

분산거래관리

보통의 접속리용과 분산거래를 위한 접속리용의 근본차이는 모든 분산거래들이 중간층에 있는 별도의 거래관리자에 의해 위탁되거나 취소된다는것이다. 따라서 응용프로그람은 거래관리자가 하는 일에 간섭할수 없다. 이것은 응용프로그람코드에서 다음과 같은 메쏘드들을 호출할수 없다는것을 의미한다.

- Connection.commit
- Connection.rollback
- Connection.setAutoCommit(true)

분산거래를 위해 만든 접속은 비분산거래들에서도 리용할수 있다.

주의: 보통의 접속에서는 자동위탁방식이 기정으로 설정되여있지만 분산거래들에 리용할수 있는 Connection객체는 자동위탁방식이 기정으로 설정되여있지 않다.

접속

Connection객체는 자료기지와의 접속을 나타낸다. 접속과정에 이루어지는 대화는 자료기지에서 실행해야 할 SQL문들과 그 접속을 통하여 귀환되는 결과들이다.

하나의 응용프로그람은 하나의 자료기지에 대한 여러개의 접속을 가질수도 있고 서로 다른 여러개의 자료기지에 대한 접속들을 가질수도 있다.

자료기지에 대한 접속을 만드는 표준적인 방법은 DataSource클라스나 DriverManager클라스에서 getConnection()메쏘드를 호출하는것이다.

사용자가 JDBC관리자층을 뛰여넘어 Driver메쏘드들을 직접 호출할수도 있다. 이것은 두개의 구동프로그람이 자료기지에 접속할수 있고 사용자가 특정한 구동프로그람을 선택하려고 하는 아주 보기드문 경우에 쓸모가 있다. 그러나 보통 DataSource클라스나 DriverManager클라스가 접속을 얻도록 하는것이 훨씬 더 쉽다.

URL(Uniform Resource Locator)은 인터네트상에서 자원을 찾기 위한 식별자이다. JDBC URL은 적당한 구동프로그람이 자료기지를 인식하고 그에 대한 접속을 확립할수 있도록 자료기지를 식별할수 있게 하는 매우 유연한 방법이다. JDBC URL은 서로 다른 구동프로그람들이 자료기지의 이름을 붙이는데서 서로 다른 방식을 리용하도록 한다. 실례로 odbc부분규약은 URL에 속성값들을 포함하도록 한다.

JDBC URL을 위한 표준문장구성법은 다음과 같다.

```
jdbc:<subprotocol>:<subname>
```

JDBC URL을 이루는 3개 부분은 다음과 같다.

- JDBC – 규약. JDBC URL에서 규약은 항상 jdbc이다.
- <subprotocol> – 하나 혹은 그 이상의 구동프로그람들에서 지원할수 있는 구동프로그람이나 접속기구의 이름.
- <subname> – 자료기지에 대한 유일한 식별자

다음의 실례는 JDBC-ODBC다리를 통하여 Customers자료기지에 접근하는 URL이다.

```
jdbc:odbc:Customers
```

odbc부분규약은 아래에 보여준것처럼 자료기지이름뒤에 필요한 속성값들을 지적할수 있다.

```
jdbc:odbc:<DSN>[;<attribute_name>=<attribute_value>]*
```

이런식으로 넘겨지는 속성들은 실례로 사용자ID라든가 통과암호일수 있다.

제4절. SQL문

일단 자료기지에 대한 접속을 확립하면 자료기지에 SQL문들을 넘겨보낼수 있다. JDBC를 리용하여 자료기지관리체계에 보낼수 있는 SQL문들의 종류에는 제한이 없으므로 사용자는 자료기지에 규정된 문들과 함께 SQL문이 아닌 문자렬들도 리용할수 있다. JDBC핵심 API는 자료기지에 SQL문들을 넘기는데 리용할수 있는 3개의 클라스들을 제공한다.

- Statement – Statement객체는 단순한 SQL문들을 보낼 때 리용되며 createStatement()메쏘드에 의해 생성된다.
- PreparedStatement – PreparedStatement는 미리 콤파일되여 PreparedStatement객체속에 저장되여있는 SQL문이다. 이 객체는 이 문을 여러번 실행할 때 리용할수 있다.
- CallableStatement – CallableStatement는 SQL저장수속들을 실행하는데 리용된다. CallableStatement는 prepareCall()메쏘드에 의해 생성된다.

3.4.1. Statement

Statement객체는 정적인 SQL문을 실행하고 그의 실행결과를 얻는데 리용된다. Statement는 SQL문을 수행하기 위한 다음의 메쏘드들을 정의한다.

- executeUpdate(String sql): 지적된 sql문을 수행하며 영향을 받은 행들의 총수 혹은 령을 돌려준다.
- executeQuery(String sql): 단일한 ResultSet를 돌려주는 SQL문을 실행한다.
- execute(String sql): 여러개의 결과들을 돌려줄수 있는 SQL문을 실행한다.

executeUpdate메쏘드는 INSERT, UPDATE, DELETE와 같은 SQL지령들을 리용할 때 영향을 받은 전체 행수를 돌려주며 CREATE TABLE과 같은 DDL지령들을 수행할 때에는 령을 돌려준다.

executeQuery메쏘드는 하나의 ResultSet를 돌려주는 SQL질문에 리용된다.

JDBC 3.0을 실현하지 않은 구동프로그람을 리용하는 경우에는 한번에 열려있는 ResultSet를 하나만 가질수 있다. 따라서 서로 다른 ResultSet로부터 자료를 엇바꾸어 얻어야 하는 경우에는 매 자료를 각이한 Statement로부터 생성해야 하며 임의의 execute메쏘드는 실행에 앞서 현재의 Resultset를 닫아야 한다. 그러나 JDBC 3.0부터는 하나의 Statement가 하나이상의 열린 ResultSet를 가질수 있다.

execute메쏘드는 여러개의 결과들을 돌려줄수 있는 SQL문을 실행하는데 리용된다. 어떤 경우에는 단 하나의 SQL문이 여러개의 ResultSet나 갱신총수를 돌려줄수도 있다. SQL문이 ResultSet를 돌려주는 경우 execute메쏘드는 참을, 갱신된 총수를 돌려주는 경우에는 거짓을 되돌린다. Statement객체는 다음과 같은 지원메쏘드들을 정의한다.

- getMoreResults
- getResultSet
- getUpdateCount

getResultSet()나 getUpdateCount()메쏘드는 결과를 검색하는데 리용하고 임의의 련속되는 결과들에로 이동하기 위해서는 getMoreResults()메쏘드를 리용한다.

3.4.2. PreparedStatement

PreparedStatement는 설정자메쏘드들을 리용하여 설정되는 IN파라메터로 알려진 변수들에 대한 자리유지자(placeholder)를 포함할수 있다. 전형적인 설정자메쏘드들은 다음과 같다.

```
public void setObject(int paramIndex, object x) throws  SQLException
```

실례로 첫번째 옹근수파라메터로 2를 설정하는 코드는 다음과 같다.

```
pstmt.setInt(1, 2);
```

PreparedStatement문의 리용실례를 목록 3-2에 보여주었다.

목록 3-2. PreparedStatement의 리용

```
package JavaDB_Bible.ch03.sec04;

import java.sql.*;

public class PreparedStmt{
  public static void main(String args[]){
    int qty;
    float cost;
    String name;
    String desc;
    String query = "SELECT * FROM Stock WHERE Item_Number = ?";
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con =
        DriverManager.getConnection("jdbc:odbc:Inventory");
      PreparedStatement pstmt = con.prepareStatement(query);
      pstmt.setInt(1, 2);

      ResultSet rs = pstmt.executeQuery();
      while (rs.next()) {
        name = rs.getString("Name");
        desc = rs.getString("Description");
        qty = rs.getInt("Qty");
        cost = rs.getFloat("Cost");
        System.out.println(name+"\t"+desc+"\t"+qty+"\t"+cost+"원");
      }
    }
    catch(ClassNotFoundException e){
      e.printStackTrace();
    }
    catch(SQLException e){
      e.printStackTrace();
    }
  }
}
```

JDBC PreparedStatement는 모든 SQL자료형들에 대하여 설정자메쏘드들을 제공한다. IN파라메터값들을 설정하기 위한 설정자메쏘드들은 입력파라메터의 SQL형과 호환할수 있는 형들을 지적하여야 한다.

setObject를 리용하면 입력파라메터를 특정한 JDBC형으로 변환할수 있다. 이 메

쏘드를 리용할 때 세번째 인수에서 변환하려는 JDBC형을 규정할수 있다. 그러면 구동프로그람은 Java객체를 자료기지에 보내기전에 지적된 JDBC형으로 변환할수 있다. JDBC형이 주어지지 않은 경우 구동프로그람은 Java객체를 기정의 JDBC형으로 넘긴다.

setByte메쏘드와 setString메쏘드는 보내는 자료의 량에 대한 제한이 없다. 또한 자바입력흐름에 IN파라메터를 설정하여 많은 량의 자료를 다룰수 있다.

JDBC는 입력흐름에 IN파라메터를 설정하는 메쏘드를 3개 가지고있다.

- setBinaryStream(해석할수 없는 바이트들을 포함하는 입력흐름용)
- setAsciiStream(ASCII문자들을 포함하는 흐름용)
- setUnicodeStream(유니코드문자를 포함하는 흐름용)

문이 실행되면 JDBC구동프로그람은 입력흐름을 반복적으로 호출하여 그의 내용들을 읽고 그것을 실제적인 파라메터값으로 자료기지에 보낸다.

setNull메쏘드는 IN파라메터로 자료기지에 NULL값을 보낼수 있게 한다. 자료기지에 NULL값을 보낼수 있는 다른 하나의 방도는 setXXX메쏘드에 NULL값을 넘기는것이다.

3.4.3. CallableStatement

CallableStatement객체는 자바프로그람에서 저장수속을 호출할수 있게 한다. 이 저장수속은 자료기지에 보관되여있으며 CallableStatement객체가 저장수속 그 자체를 포함하지는 않는다. 목록 3-3의 실례는 저장수속의 생성과 리용을 보여준다.

목록 3-3. 저장수속의 생성과 리용

```
package JavaDB_Bible.ch03.sec04;

import java.sql.*;

public class CallableStmt{
  public static void main(String args[]){
    int orderNo;
    String name;
    String storedProc = "CREATE PROCEDURE SHOW_ORDERS_BY_STATE "+
                        "@State CHAR(8) AS "+
                        "SELECT c.Family_Name+c.Personal_Name AS Name,"+
                        "o.Order_Number "+
                        "FROM Customers c, Orders o "+
                        "WHERE c.Customer_ID =o.Customer_ID " +
                        "AND c.State = @State " +
```

```
                              "ORDER BY c.Family_Name;";
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:Customers");
      Statement stmt = con.createStatement();
      stmt.executeUpdate(storedProc);

      CallableStatement cs = con.prepareCall("{call SHOW_ORDERS_BY_STATE(?)}");
      cs.setString (1, "평안남도");
      ResultSet rs = cs.executeQuery();

      while (rs.next()) {
        name = rs.getString("Name");
        orderNo = rs.getInt ("Order_Number");
        System.out.println(name + " : "+orderNo);
      }
    }
    catch(ClassNotFoundException e){
      e.printStackTrace();
    }
    catch(SQLException e){
      e.printStackTrace();
    }
  }
}
```

저장수속호출에서 JDBC탈출문자렬이 리용된다는데 대하여 주의하기 바란다. 이것은 저장수속이 모든 자료기지관리체계들에 대하여 표준적인 방법으로 호출될수 있게 한다. CallableStatement는 PreparedStatement의 확장이므로 입력파라메터들을 취할수 있다. 또한 출력파라메터 혹은 입력파라메터와 출력파라메터 둘다 가질수 있다.

실례에서 보여준것처럼 물음기호(?)는 @Name약속을 리용하는 저장수속에서 정의한 파라메터들에 대한 자리유지자로 된다. IN파라메터값들은 PreparedStatment로부터 계승된 설정자메쏘드들을 리용하여 설정된다. 만일 출력파라메터를 사용하였다면 그것은 execute메쏘드들이 어느 하나라도 호출되기전에 registerOutParameter()메쏘드를 리용하여 OUT파라메터로 등록되여야 한다. 다음의 실례를 보자.

```
cstmt.registerOutParameter(1, java.sql.Types.VARCHAR);
```

OUT파라메터값들은 그 자료형에 알맞는 취득자메쏘드들을 실행한 다음 추출할수 있다. 목록 3-4는 자료기지에 대한 사용자이름과 통과암호을 대조하여 유효한 사용자이면 문자렬 "PASS"를 돌려주고 아니면 "FAIL"을 돌려주는 간단한 저장수속을 보여준다.

목록 3-4. 입출력파라메터를 가진 저장수속

```
CREATE PROCEDURE CHECK_USER_NAME
  @FName VARCHAR(30),
  @PName VARCHAR(20),
  @PassFail VARCHAR(20) OUTPUT
As
  IF EXISTS (SELECT * FROM Customers
               WHERE Family_Name = @FName AND Personal_Name = @PName)
    SELECT @PassFail = "PASS"
  ELSE
    SELECT @PassFail = "FAIL"
```

일부 관계형자료기지관리체계들에서 제기되는 제한성으로 하여 CallableStatement객체의 실행에 의하여 생성되는 모든 결과들은 OUT파라메터들을 얻기전에 검색되여야 한다. 즉 CallableStatement객체가 여러개의 ResultSet객체들을 돌려주는 경우 모든 결과들은 OUT파라메터를 얻기전에 getMoreResults메쏘드들에 의해 검색되여야 한다.

목록 3-5에서는 JDBC응용프로그람에서 저장수속으로부터 출력파라메터를 어떻게 검색해야 하는가를 보여준다.

목록 3-5. 저장수속으로부터 출력파라메터 얻기

```
package JavaDB_Bible.ch03.sec04;

import java.sql.*;

public class CheckPassword{
  public static void main(String args[]){
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:Customers");

      CallableStatement cs =con.prepareCall("{call CHECK_USER_NAME(?,?,?)}");
        cs.setString(1, "김");
        cs.setString(2, "성철");
        cs.registerOutParameter(3, java.sql.Types.VARCHAR);
        cs.executeUpdate();
        System.out.println(cs.getString(3));
    }
    catch(ClassNotFoundException e) {
```

```
      e.printStackTrace();
    }
    catch(SQLException e){
      e.printStackTrace();
    }
  }
}
```

실천에서는 SQL문들의 묶음이 전체로서 실행되는것이 중요하다. 대표적인 실례는 당좌예금에서 저축예금에로 자금을 이동하는 경우이다. 은행의 관점에서 어떤 리유로 체계가 첫 구좌의 차방기입에 실패하는 경우 두번째 구좌의 대방기입이 일어나지 말아야 한다. 단일한 거래로서 일련의 SQL문들을 관리하는 문제는 다음 체계에서 론의한다.

제5절. 거래와 일괄갱신

여러 SQL문들이 하나의 실체로서 실행되도록 묶는 능력은 SQL의 거래방식을 통하여 제공된다. 거래는 완전하게 실행되여야 할 하나 혹은 여러개의 문들로 구성되며 집체적으로 위탁되든지 본래의 상태로 되돌아가야 간다. 자료기지거래의 문맥에서 **위탁**(commit)이라는 말은 자료기지에서 《영구적인》 변화가 이루어진다는것을 의미하며 **되돌리기**(roll back)는 자료기지에서 아무런 변화도 일어나지 않는다는것을 의미한다. commit 혹은 rollback메쏘드가 호출되면 현재의 거래는 끝나고 다른 거래가 시작된다.

새로운 JDBC접속은 기정으로 자동위탁방식으로 된다. **자동위탁방식**(auto-commit)이란 문이 실행될 때 commit메쏘드가 자동적으로 호출된다는것을 의미한다. 위탁은 문을 완료하거나 다음 실행이 발생할 때 일어난다. ResultSet를 돌려주는 문들의 경우에 문은 ResultSet의 마지막 행이 검색되거나 ResultSet가 닫길 때 완료된다. 하나의 문이 출력파라메터값들은 물론 여러개의 결과들을 돌려줄수 있다. 여기서 위탁은 모든 결과들과 출력파라메터값들이 검색되였을 때 일어난다.

자동위탁방식은 다음의 방법으로 조종된다.

```
public void setAutoCommit(boolean autoCommit) throws SQLException
```

자동위탁방식이 불가능으로 된 경우 거래는 commit메쏘드나 rollback메쏘드가 명시적으로 호출되기전까지 완료되지 못하며 따라서 그것은 commit 혹은 rollback메쏘드를 마지막으로 호출한 때로부터 실행된 모든 문들을 포함하고있다. 이 경우 거래에 속한 모든 문들은 집단적으로 위탁되거나 취소된다.

SQL문들이 자료기지에 변화를 가져올 때 commit메쏘드는 이 변화들을 확정적인것으로 만들고 임의의 닫겨진 거래유지를 해방한다. 한편 rollback메쏘드는 이 변화들을 거부한다.

명백히 은행들사이 자금이동과 같은 상태에서는 자동위탁을 불가능으로 하고 두가지 갱신을 하나의 거래로 묶어 어느 한 쪽이 실패하는 경우에 초래되는 사고를 막을수 있다. 두 갱신이 다 성공하는 경우에는 commit메쏘드가 호출되여 두 갱신의 효과를 확정적으로 고착하고 어느 한쪽이라도 실패하는 경우에는 rollback메쏘드가 호출되여 갱신이 이루어지기전 구좌의 잔고상태를 회복한다.

대부분의 JDBC구동프로그람들은 거래를 지원한다. DatabaseMetaData는 자료기지관리체계가 제공하는 거래지원의 수준을 서술하는 정보를 준다.

3.5.1. 거래격리준위

거래관리의 기본개념은 련결된 몇개의 SQL지령들을 하나로 수행하는데서 실패하는 경우에 일어나게 되는 불일치를 관리할수 있게 한다. 그러나 관리의 추가적인 준위를 요구하는 다른 형태의 불일치가 일어날수 있다.

례를 들어 다중사용자 응용프로그람에서 한 거래가 두 구좌들사이의 자금이동을 시작하고 아직 그것을 위탁하지는 않았는데 다른 사용자에 의한 두번째 거래가 문제의 그 구좌들중 어느 하나에 접근하려고 시도하는 경우가 있을수 있다. 만일 첫 거래가 취소되는 경우 두번째 거래가 읽은 값은 정당한것이 못된다. 어떤 경우에는 이것이 허용될수도 있겠지만 재정관리부문에서는 이것을 용납할수 없다.

JDBC는 각이한 불일치를 관리하는 5가지 준위의 거래격리를 정의한다. 가장 낮은 준위는 거래들이 전혀 지원되지 않는것을 말하며 기타는 SQL-92가 정의하는 4개의 격리준위들과 일치한다.

- 위탁전 읽기(Read Uncommitted)
- 위탁후 읽기(Read Committed)
- 반복 읽기(Repeatable Read)
- 련쇄가능(serializable)

가장 높은 격리준위는 한 거래가 자료기지를 조작하는 동안 다른 거래들은 그 거래가 읽는 자료에 아무런 변화도 줄수 없다는것을 지적한다.

SQL-92격리준위들은 현상(phenomena)이라고 부르는 3가지 금지된 조작들에 의하여 규정된다.

- 불결한 읽기: 이것은 한 거래가 다른 거래의 작용이 위탁되기전에 그 결과를 볼수 있는 경우에 일어난다.
- 반복불가능 읽기(혹은 모호읽기): 이것은 한 거래의 결과들이 위탁되기전에 다른 거래에 의해 변경 혹은 삭제될수 있는 경우에 일어난다.
- 착각읽기: 이것은 한 거래에서 한 질문의 결과들이 위탁되기전에 다른 거래에 의해 변화될수 있는 경우에 일어난다.

SQL-92격리준위들은 표 3-1에서 보여준것처럼 이 현상들중 어느것이 주어진 격리준위에서 일어날수 있는가로 정의된다.

표 3-1. SQL-92격리준위

격리준위	불결한 읽기	반복불가능읽기	착각읽기
위탁전 읽기	예	예	예
위탁후 읽기	아니	예	예
반복가능읽기	아니	아니	예
련쇄가능(serializable)	아니	아니	아니

표 3-1로부터 위탁후 읽기가 지원되는 경우 불결한 읽기는 일어나지 않지만 반복불가능읽기 혹은 착각읽기가 일어날수 있다. 류사하게 반복읽기가 지원되는 경우 불결한 읽기나 반복불가능읽기는 일어나지 않지만 착각읽기는 일어날수 있다.

보통 격리준위가 높을수록 격리되는 시간이 길고 사용자들사이의 동시작용이 적으므로 응용프로그람실행이 느리다. 이것은 어떤 격리준위를 리용할것인가를 결심할 때 성능과 자료일관성사이의 타협이 이루어져야 한다는것을 의미한다.

현재의 격리준위는 다음의 메쏘드를 리용하여 알아볼수 있다.

```
public int getTransactionIsolation();
```

이 메쏘드는 다음과 같은 격리준위코드를 돌려준다.

- TRANSACTION_NONE: 거래가 지원되지 않는다.
- TRANSACTION READ_COMMITTED: 불결한 읽기를 막고 반복불가능 읽기와 착각읽기가 일어날수 있다.
- TRANSACTION_READ_UNCOMMITTED: 불결한 읽기, 반복불가능읽기, 착각읽기가 발생할수 있다.
- TRANSACTION_REPEATABLE_READ: 불결한 읽기와 반복불가능읽기는 일어나지 않고 착각읽기가 일어날수 있다.

- TRANSACTION_SERIALIZABLE: 불결한 읽기, 반복불가능 읽기, 착각읽기가 다 일어나지 않는다.

접속의 격리준위는 다음의 메쏘드에 의해 통제된다.

```
con.setTransactionIsolation(TRANSACTION_ISOLATION_LEVEL_XXX);
```

례를 들어 다음의 코드로 자료기지관리체계에 위탁전 읽기를 지시할수 있다.

```
con.setTransactionIsolation(TRANSACTION_READ_UNCOMMITTED);
```

새로운 Connection객체를 생성할 때 그의 거래격리준위는 보통 자료기지에 해당한 기정준위로 설정된다. 거래격리준위를 변화시키기 위해서는 setIsolationLevel메쏘드를 호출한다.

거래를 시작하기전에 거래격리준위를 설정하고 거래가 끝난 다음 그것을 재설정하여 거래마다에 거래격리준위를 변화시킬수도 있다.

주의: 거래처리기간에는 거래격리준위를 변화시키지 않는것이 좋다. 그것은 이때 commit메쏘드가 즉시 호출되고 변하지 말아야 할 그 시점까지 변화들이 일어날수 있기때문이다.

3.5.2. 거래보관점

거래보관점은 거래의 위탁과 취소를 보다 훌륭히 조종할수 있도록 JDBC 3.0에서 강화된 부분이다. **거래보관점**(savepoint)이란 거래기간에 거래처리를 위한 조작들사이에 표식자처럼 삽입되여 거래가 취소되는 경우 전체의 취소가 아니라 바로 표식자까지만 되돌아가고 표식자이전의 모든 조작들은 그대로 유지되게 하는 거래의 완충점을 말한다.

다음의 실례에서는 첫번째 갱신후에 거래보관점을 설정하고있다. 거래가 취소될 때 Savepoint1까지만 되돌아가기때문에 두번째, 세번째 갱신들은 무효로 되지만 첫번째 갱신은 그대로 남아있게 된다.(인수 update1, update2, update3은 SQL지령들을 나타낸다.)

```
con.setAutoCommit(false);
Statement stmt = con.createStatement();

stmt.executeUpdate(update1);

Savepoint savePoint1 = con.setSavepoint("SavePoint1");

stmt.executeUpdate(update2};
```

```
stmt.exeeuteUpdate(update3);

con.rollback(savePoint1);
con.commit();
```

3.5.3. 다중스레드

JDBC명세들은 모든 java.sql객체들에 대한 조작들이 안전하게 스레드될것을 요구한다. 이것은 JDBC가 여러 스레드가 같은 객체를 동시에 호출하는 경우에 대처할수 있어야 한다는것을 의미한다. 일부 구동프로그람들은 이것을 완전한 동시발생으로 제공할수 있으며 다른 구동프로그람들은 어떤 스레드가 하나의 문을 실행하면 그 실행이 완료될 때까지 다음 스레드를 보내지 않고 기다린다.

주의: 다중스레드를 리용하는 한가지 특수한 방법은 오래동안 실행되고있는 문을 중지하는것이다. 이것은 그 문을 실행하는 하나의 스레드를 새로 만들고 자기의 Statement.cancel()메쏘드로 그 문을 중지하는데 다른 스레드를 리용할수 있다는것을 의미한다.

3.5.4. 일괄갱신

일괄갱신(batch update)은 자료기지에 대한 조작을 일괄로 처리하기 위한 갱신문들의 모임이다. 이것은 갱신문들을 개별적으로 보내는것보다 훨씬 더 효과적이다. 일괄갱신은JDBC 2.0 API에서부터 지원된다. JDBC 1.0 API에서는 여러개의 갱신문들이 같은 거래의 부분이라고 해도 자료기지에 개별적으로 제기되고 개별적으로 처리된다.

JDBC 2.0 API에서 Statement, PreparedStatement, CallableStatement는 행의 갱신과 삽입, 삭제를 위한 문들을 포함할수 있는 일괄목록들을 지원한다. 일괄목록들에는 CREATE TABLE과 DROP TABLE과 같은 DDL문들도 포함할수 있다.

주의: 일괄갱신에는 갱신개수를 알수 있는 문들만이 리용될수 있다. SELECT문과 같이 ResultSet객체를 돌려주는 문들은 일괄갱신에 리용될수 없다.

일괄갱신관리에 리용되는 지령들은 다음과 같다.

- addBatch: SQL지령들을 일괄목록에 추가한다.
- clearBatch: 일괄목록을 비운다.
- executeBatch: 목록안에 있는 모든 문들을 묶음으로 실행한다.

새로운 행을 일괄목록에 삽입하는 코드는 다음과 같다.

```
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO CUSTOMERS VALUES('최강호')");
stmt.addBatch("INSERT INTO CUSTOMERS VALUES('최강철')");
stmt.addBatch("INSERT INTO CUSTOMERS VALUES('최영심')");

int[] updateCounts = stmt.executeBatch();
con.commit();
con.setAutoCommit(true);
```

자료기지관리체계는 지령들을 일괄목록에 추가된 순서대로 실행하며 갱신개수의 옹근수배렬을 돌려준다. 갱신개수의 배렬은 일괄목록중 성과적으로 실행된 지령결과들을 실행된 순서대로 나타낸다.

지령묶음에서 임의의 지령들을 성과적으로 실행할수 없는 경우 BatchUpdateException이 발생한다.

갱신개수배렬이 성공적으로 실행된 지령들의 결과를 나타내기때문에 귀환된 배렬의 길이로부터 실행된 지령개수를 쉽게 식별할수 있다.

주의: 일괄갱신중에 오유가 발생하는 경우 그것들은 적당히 처리될수 있기때문에 일괄갱신기간에 자동위탁방식은 항상 불가능으로 해야 한다. 우의 실례에서 보여준것처럼 갱신을 위탁하기 위해서는 특정한 commit()지령을 수행해야 한다.

BatchUpdateException은 SQLExcepton를 확장하였으며 executeBatch메쏘드가 돌려주는 배렬과 류사한 갱신개수의 배렬을 추가한다. 아래에 보여준것처럼 getUpdateCounts()를 리용하여 이 배렬을 검색할수 있다.

```
try {
   //...
}
catch(BatchUpdateException b) {
   System.err.print("Update counts: ");
   int[] updateCounts = b.getUpdateCounts();
   for (int i = 0; i < updateCounts.length; i++) {
      System.err.println(updateCounts[i]);
   }
}
```

갱신개수들은 일괄목록에 추가된 지령들과 같은 순서이므로 묶음에서 어느 지령이 성공적으로 실행되였는가를 나타낼수 있다.

SQL질문에 의해 돌아온 결과들은 java.sql.ResultSet객체에 보관된다. 이 객체들은 다음 절에서 론한다.

제6절. 결과모임

ResultSet는 SQL질문이 돌려주는 자료로서 질문의 조건을 만족시키는 모든 행들을 포함하고있다. ResultSet는 질문의 조건들을 만족시키면서 문에서 지적된 렬순서대로 값들을 현시하는 표처럼 생각할수 있다. 례를 들어 다음과 같은 질문의 경우 결과모임은 표 3-2와 같다.

```
SELECT Name, Description, Qty, Cost FROM Stock
```

표 3-2. ResultSet의 결과

Name	Description	Qty	Cost
랭천사이다	청량음료	23	2.5원
평양맥주	청량음료	43	3.4원
모란과자	당과류	20	1.5원

ResultSet는 ResultSet의 취득자메쏘드를 통하여 접근할수 있는 자료의 행을 지적하는 유표를 유지한다. ResultSet.next()메쏘드가 호출될 때마다 유표는 한행씩 밑으로 이동한다.

ResultSet가 처음 생성되였을 때 유표는 첫행 전에 위치한다. 따라서 ResultSet를 생성한 다음에 유표가 첫행에 놓이자면 next()메쏘드를 호출해야 한다. 지적한 자료행이 정당한 경우 next()는 참을 돌려주기때문에 목록 3-6에 보여준것처럼 행자료에 접근하기 위하여 while순환을 리용할수 있다.

목록 3-6. 결과모임의 검색

```
package JavaDB_Bible.ch03.sec07;

import java.sql.*;

public class PrintResultSet{
  public static void main(String args[]){
    String query = "SELECT Name, Description, Qty, Cost FROM Stock";
    PrintResultSet p = new PrintResultSet(query);
  }

  public PrintResultSet(String query){
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection("jdbc:odbc:Inventory");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      System.out.println("Name\tDescription\tQty\tCost");

      while (rs.next()) {
        System.out.print(rs.getString("Name")+"\t");
        System.out.print(rs.getString("Description")+"\t");
        System.out.print(rs.getInt("Qty")+"\t");
        System.out.println(rs.getFloat("Cost")+"원");
      }
    }
    catch(ClassNotFoundException e){
      e.printStackTrace();
    }
    catch(SQLException e){
      e.printStackTrace();
    }
  }
}
```

next()호출에 의하여 유표가 한행씩 밑으로 이동하는데 따라 ResultSet의 행들은 제일 꼭대기행부터 내려가면서 차례로 검색된다. 유표는 ResultSet객체나 그것을 생성한 Statement객체가 닫길 때까지 유효하다.

자료는 그 자료를 포함하는 렬을 참조하는 취득자메쏘드들을 리용하여 ResultSet로부터 검색된다. ResultSet취득자메쏘드들은 현재행으로부터 렬값들에 대한 자료형 특정의 검색을 제공한다. 매개 행에서 렬값들은 임의의 순서로 검색될수 있다.

ResultSet객체가 제공하는 취득자메쏘드들을 표 3-3에 보여주었다. 매 취득자메쏘드는 렬참조와 관련하여 두가지 변종을 가지는데 하나는 이름에 의한 렬참조이고 다른 하나는 렬번호에 의한 참조이다.

표 3-3. ResultSet의 취득자메쏘드들

자료형	메쏘드
BigDecimal	getBigDecimal(String columnName, int scale)
boolean	getBoolean(String columnName)
byte	getByte(String columnName)
byte[]	getBytes(String columnName)
double	getDouble(String columnName)
float	getFloat(String columnName)
int	getInt(String columnName)
java.io.InputStream	getAsciiStream(String columnName)
java.io.InputStream	getUnicodeStream(String columnName)
java.io.InputStream	getBinaryStream(String columnName)
java.sql.Date	getDate(String columnName)
java.sql.Time	getTime(String columnName)
java.sql.Timestamp	getTimestamp(String columnName)
long	getLong(String columnName)
Object	getObject(String columnName)
short	getShort(String columnName)
String	getString(String columnName)

주의: 렬번호는 0부터가 아니라 1부터 시작한다.

자료는 렬이름으로 검색할수도 있고 렬번호에 의해 검색할수도 있다. 레를 들어 앞의 실례들은 렬이름을 리용하였다. 그러나 렬번호를 아는 경우에는 렬번호를 리용하여 같은 결과를 얻을수 있다.

```
ResultSet rs = stmt.executeQuery("SELECT Family_Name, Personal_Name FROM
Customers");
while(rs.next()) {
   System.out.println(rs.getString(1) + rs.getString(2));
}
```

취득자메쏘드에서 렬이름을 리용하는것은 사용자가 질문에서 렬이름을 지적하는 경우 취득자메쏘드의 인수로서 같은 이름을 리용할수 있게 하기 위해서이다.

렬이름들을 리용하는데서 일어날수 있는 문제는 두개의 표를 결합하는 경우 같은 이름을 가진 하나이상의 렬을 가지는 결과모임을 돌려주는 SQL문이 있을수 있다는것이다. 그때 취득자메쏘드의 파라메터로 렬이름을 리용하면 처음으로 일치하는 렬이름의 값을 돌려주게 된다.

Oracle과 같은 일부 자료기지관리체계들에서는 이 문제를 해결하기 위하여 "table_name.column_name"형식의 완전한 렬이름들을 리용한다. 그러나 MSAccess는 같은 이름을 가진 여러개의 렬이 있는 경우 렬색인을 리용한다.

3.6.1. 흘리기가능한 결과모임

JDBC 2.0 API에 추가된 기능들중의 하나가 유표를 앞뒤로 자유롭게 이동시킬수 있는 ScrollableResultSet이다. 유표를 앞뒤로 이동시키는 메쏘드와 함께 유표의 현재위치를 얻고 특정한 행에로 단번에 이동시키기 위한 메쏘드들도 있다.

흘리기가능한 결과모임의 작성

java.sql.Statement객체가 돌려주는 ResultSet의 류형은 Statement가 창조될 때 정의된다.

Statement를 창조하는 Connection의 createStatement메쏘드에는 두가지 형식이 있다.

```
public Statement createStatement() throws SQLException
public Statement createStatement(int rsType, int rsConcurrency)
   throws SQLException
```

첫번째 메쏘드는 앞에서 본 실례들에서 이미 고찰하였다.

두번째 메쏘드는 흘리기가능하고 갱신가능한 ResultSet들을 생성할수 있게 한다. 첫번째 인수 rsType는 ResultSet객체의 형을 지적하는 상수로서 다음 3가지중의 하나를 취해야 한다.

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

rsType로 TYPE_FORWARD_ONLY를 규정하면 흘리기할수 없는 결과모임 즉 유표가 앞방향으로만 이동하는 결과모임을 생성한다. 만일 이때 두번째 인수에 CONCUR_READ_ONLY를 지적하는 경우 인수가 없는 메쏘드에서 생성되는 ResultSet와 꼭 같은 기정의 ResultSet를 얻게 된다.

흘리기가능한 ResultSet객체를 얻기 위해서는 TYPE_SCROLL_INSENSITIVE나 TYPE_SCROLL_SENSITIVE를 지적하여야 한다. TYPE_SCROLL_INSENSITIVE를 리용하면 정의된 결과모임이 열려있는 동안에 일어난 변화를 반영하지 않지만 TYPE_SCROLL_SENSITIVE를 리용하면 그 변화를 반영한다. 이것이 흘리기가능한 두가지 결과모임의 차이점이다. 물론 사용자가 결과모임을 닫고 그것을 다시 열면 결과모임의 류형에 관계없이 항상 변화된 내용들을 볼수 있다.

두번째 인수는 결과모임에 대하여 읽을수만 있는가 혹은 갱신도 가능한가를 규정하는 상수이다. 이 ResultSet상수는 CONCUR_READ_ONLY와 CONCUR_UPDATABLE이다. 사용자가 결과모임의 류형을 지적하는 경우에는 역시 결과모임이 읽기만 가능한것인가 갱신가능한것인가를 지적하여야 한다.

ResultSet.getType()메쏘드는 ResultSet객체가 흘리기가능한가 혹은 읽기만 가능한가를 검사한다.

```
if( rs.getType() == ResultSet.TYPE_FORWARD_ONLY )
   System.out.println("FORWARD_ONLY");
else
   System.out.println("SCROLLABLE");
```

유표조종

일단 사용자가 흘리기가능한 ResultSet객체를 얻게 되면 그 결과모임에서 유표를 마음대로 움직일수 있다. 흘리기가능한 ResultSet에는 유표를 한행씩 전진시키는 next()메쏘드와 유표를 한행씩 후진시키는 previous()메쏘드가 있다.

이 두 메쏘드들은 유표가 결과모임의 범위를 넘어서면(즉 유표가 마지막행의 뒤로 넘어 전진하거나 첫행의 앞으로 후진하는 현상) 거짓을 돌려주기때문에 while순환에서 리용할수 있다. 목록 3-7은 목록 4-6의 기정 ResultSet를 흘리기가능한 ResultSet로 바꾸었다.

목록 3-7. 흘리기가능한 ResultSet

```
public void PrintResultSet(String query){
  try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection("jdbc:odbc:Inventory");
    Statement stmt = con.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_READ_ONLY);
    ResultSet rs = stmt.executeQuery(query);
    ResultSetMetaData md = rs.getMetaData();

    int nColumns = md.getColumnCount();
    for(int i=1;i<=nColumns;i++){
      System.out.print(md.getColumnLabel(i)
+((i==nColumns)?"\n":"\t"));
      if(i==2)System.out.print("\t");
    }
    while (rs.next()) {
      for(int i=1; i<=nColumns; i++){
         System.out.print(rs.getString(i)+((i==nColumns)? "\n":"\t"));
      }
    }
    while (rs.previous()) {
      for(int i=1; i<=nColumns; i++){
        System.out.print(rs.getString(i)+((i==nColumns)?" \n":"\t"));
      }
    }
  }
  catch(ClassNotFoundException e) {
    e.printStackTrace();
  }
  catch(SQLException e){
    e.printStackTrace();
  }
}
```

실례에서는 우선 첫 3개행을 인쇄하고 다시 그것들을 반대순서로 인쇄한다.

Name	Description	Qty	Cost
랭천사이다	청량음료	23	2.5
평양맥주	청량음료	43	3.4
모란과자	당과류	20	1.5
모란과자	당과류	20	1.5
평양맥주	청량음료	43	3.4
랭천사이다	청량음료	23	2.5

유표의 도약이동

유표를 전진 및 후진시키는 next()메쏘드와 previous()메쏘드외에도 다음과 같은 유표이동메쏘드들이 있다.

- first()
- last()
- beforeFirst()
- afterLast()
- absolute(int rowNumber)
- relative(int rowNumber)

처음 4개 메쏘드들의 효과는 이름으로부터 명백하다.

absolute()메쏘드는 인수에서 지적한 행번호로 유표를 이동한다. 만일 번호가 정수이면 첫 행에서부터 시작하여 지적된 행번호에로 이동하고 부수이면 마지막 행에서부터 반대로 거슬러 지적된 행번호로 이동한다.

주의: 행번호는 1부터 계수하기때문에 absolute(1)은 유표를 첫 행에, absolute(-1)은 유표를 마지막 행에 놓는다.

relative(int rowNumber)메쏘드는 현재 유표가 있는 행으로부터 정해진 방향으로 얼마만한 행을 이동하는가를 지정한다. 인수가 정수이면 유표를 주어진 행만큼 앞방향으로, 부수이면 유표를 주어진 행만큼 뒤방향으로 이동시킨다.

유표의 위치얻기

다음의 메쏘드들은 유표의 현재위치를 얻을수 있게 해준다.

- isFirst()
- isLast()
- isBeforeFirst()
- isAfterLast()
- getRow()

이 메쏘드들의 작용은 이름으로부터 명백하므로 더 설명하지 않는다.

주의: isAfterLast()메쏘드는 유표가 마지막행 다음에 있지 않을 때에는 물론 결과모임이 비여있을 때에도 거짓을 돌려주므로 isAfterLast()메쏘드로부터 귀환된 false값은 자료가 유용한가를 알아보는데 리용할수 없다.

3.6.2. 갱신가능한 결과모임

UpdatableResultSet는 이름그대로 갱신가능한 결과모임이다. 사용자들은 ResultSet자체에서 값들을 갱신할수 있으며 그 변화들은 자료기지에 반영된다.

UpdatableResultSet객체를 생성하려면 createStatement메쏘드를 호출할 때 두번째 인수로서 상수 CONCUR_UPDATABLE를 지적하여야 한다. 그러면 Statement객체는 질문을 수행할 때 갱신가능한 ResultSet객체를 만든다.

주의: 갱신가능한 ResultSet객체가 반드시 흘리기가능한것은 아니다.

일단 UpdatableResultSet객체를 가지면 새로운 행을 삽입하고 이미 있는 행을 삭제할수 있으며 렬값들을 변경할수 있다.

경고: 갱신가능한 결과모임을 얻기 위한 질문은 일반적으로 기본열쇠를 선택된 렬들중의 하나로 지적해야 하며 따라서 렬들은 하나의 표에서만 선택되여야 한다.

결과모임이 갱신가능하다는것은 사용자가 얻은 결과모임이 실제적으로 갱신가능하게 될것이라는것을 담보하지는 않는다. UpdatableResultSet의 요구는 사용하고있는 구동프로그람에 의존하며 갱신가능한 결과모임을 지원하지 않는 구동프로그람들은 읽기만 가능한 결과모임을 돌려준다. 그러므로 ResultSet.getConcurrency()를 리용하여 ResultSet가 갱신가능한것인가 아닌가를 확인해야 한다. 목록 3-8은 흘리기가능하고 갱신가능한 ResultSet을 열고 그것이 실제로 갱신가능한가를 알아보는데 getConcurrency를 리용하고있다.

목록 3-8. 갱신가능한 결과모임의 열기

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:Customers");
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData md = rs.getMetaData();

if(rs.getConcurrency() == ResultSet.CONCUR_UPDATABLE)
  System.out.println("UPDATABLE");
else
  System.out.println("READ_ONLY");

int nColumns = md.getColumnCount();
for(int i=1; i<=nColumns; i++){
  System.out.print(md.getColumnLabel(i)+((i==nColumns)?"\n":"\t"));
}
while (rs.next()) {
  rs.updateString("State", "평안남도");
  rs.updateRow();
  for(int i=1;i<=nColumns;i++){
    System.out.print(rs.getString(i)+((i==nColumns)?"\n":"\t"));
  }
}
```

만일 구동프로그람이 UpdatableResultSet의 정의를 지원하지 않는다면 Statement객체는 SQL례외 "Opional feature not implemented"를 내보낼수 있다.

결과모임의 갱신

UPDATE지령에 비해 UpdatableResultSet를 리용하는 방법은 아주 간단하다. 사용자는 요구되는 행에 유표를 설정하고 자료형에 따르는 갱신메쏘드를 리용하여 렬값을 변화시킨다. 아래에 그 실례를 보여준다.

```
rs.updateString("State", "평안남도");
```

렬값의 변화는 항상 현재 행에서 진행되기때문에 갱신전에 반드시 갱신하려는 행으로 유표를 이동하여야 한다.

대부분의 갱신메쏘드들은 두개의 파라메터 즉 갱신하려는 렬과 그 렬에 넣을 새로운 값을 가진다. 표 3-4에서는 UpdatableResultSet의 메쏘드들을 보여준다.

주의: 특별한 갱신메쏘드인 updateNull()은 렬값을 NULL로 설정하는데 리용된다.

표 3-4. 결과모임갱신메쏘드

자료형	메쏘드
BigDecimal	updateBigDecimal(String columnName, BigDecimal x)
boolean	updateBoolean(String columnName, boolean x)
byte	updateByte(String columnName, byte x)
Byte[]	updateBytes(String columnName, byte[] x)
double	updateDouble(String columnName, double x)
float	updateFloat(String columnName, float x)
int	updateInt(String columnName, int x)
java.io.InputStream	updateAsciiStream(String columnName, InputStream x, int length)
java.io.InputStream	updateUnicodeStream(String columnName, InputStream x, int length)
java.io.InputStream	updateBinaryStream(String columnName, InputStream x, int length)
java.sql.Date	updateDate(String columnName, Date x)
java.sql.Time	updateTime(String columnName, Time x)
java.sql.Timestamp	updateTimestamp(String columnName, Timestamp x)
long	updateLong(String columnName, long x)
Object	updateObject(String columnName, Object x)
Object	updateObject(String columnName, Object x, int scale)
short	updateShort(String columnName, short x)
String	updateString(String columnName, String x)
NULL	updateNull(String columnName)

ResultSet에서 렬값들을 갱신한 다음에는 유표를 이동하기전에 자료기지에서의 최종적인 변화를 위하여 반드시 ResultSet의 updateRow()메쏘드를 호출하여야 한다. 갱신메쏘드들을 리용한 변화들은 updateRow()가 호출될 때까지 반영되지 않는다.

주의: updateRow()를 호출하기전에 유표를 다른 행에로 이동하면 갱신값들이 사라지고 행은 자기의 이전 렬 값들로 되돌아간다.

updateRow()를 호출하기전에 갱신을 취소하려면 cancelRowUpdates()메쏘드를 호출한다. 그러나 일단 updateRow()가 호출되면 cancelRowUpdates()는 더이상 작용하지 못한다.

새로운 행의 삽입

UpdatableResultSet는 개별적인 자료마당의 갱신외에 완전한 행의 삽입과 삭제도 지원한다.

ResultSet객체는 삽입행(insert row)을 가지고있으며 이 행은 새로운 행을 삽입하기 위한 완충기로서의 역할을 한다.

새 행은 앞에서 론의한 행의 갱신과 류사한 방법으로 작성된다.

1. moveToInsertRow()메쏘드를 호출하여 삽입행으로 유표를 이동시킨다.
2. 적당한 갱신메쏘드를 리용하여 그 행의 매 렬에 새로운 값을 설정한다.
3. 새로운 행을 결과모임에 삽입함과 동시에 자료기지에 삽입하기 위하여 insertRow()메쏘드를 호출한다.

목록 3-9는 새로운 행을 자료기지에 삽입하기 위한 UpdatableResultSet의 리용을 보여주고있다.

목록 3-9. 새로운 행삽입을 위한 UpdatableResultSet의 리용

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:Customers");
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(query);
rs.moveToInsertRow();

rs.updateInt("Customer_ID", 150);
rs.updateString("Family_Name", "정");
rs.updateString("Personal_Name", "영식");

rs.insertRow();
```

행의 매개 렬들에 값을 다 주지 않고 새로운 행을 삽입하는 경우 빈 렬들에 대하여

기정값이 있다면 그 값이 리용된다. 그밖에 그 렬이 NULL값을 허용한다면 NULL이 삽입된다. 이것도 저것도 다 아닌 경우에는 SQLException이 발생한다.

갱신을 요구하여 지적한 렬이 행을 삽입하려는 ResultSet에 없는 경우에도 SQLException이 발생한다. 때문에 ResultSet객체를 얻기 위한 질문에서는 SELECT문에 의해 귀환되는 렬의 수를 제한하기 위하여 WHERE문절을 리용하지 않고 모든 렬들을 다 선택하는것이 보통이다.

주의: insertRow()를 호출하기전에 삽입행으로부터 유표를 이동하면 삽입행에 추가된 모든 값들이 분실된다.

삽입행으로부터 결과모임에로 되돌아가려면 first(), last(), beforeFirst(), afterLast(), absolute()와 같은 메쏘드들을 리용한다. 더우기 유표가 삽입행에 있을때에만 호출할수 있는 특수한 메쏘드 moveToCurrentRow()를 리용할수도 있다. 이 메쏘드는 삽입행으로 이동하기 바로 전에 현재행이였던 행으로 유표를 돌려보낸다. 결과모임은 삽입행에 대한 접근이 진행되는 동안 삽입행으로 이동하기전 현재행의 레코드를 유지하기때문에 previous메쏘드와 relative메쏘드를 리용할수도 있다.

행의 삭제

UpdatableResultSet에서 행의 삭제는 매우 간단하다. 삭제하려는 행에 유표를 이동시키고 deleteRow메쏘드를 호출한다. 다음의 실례에서는 ResultSet객체를 얻고 거기서 세번째 행을 삭제하고있다.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con =
   DriverManager.getConnection("jdbc:odbc:Customers");
Statement stmt = con.createStatement(
   ResultSet.TYPE_SCROLL_INSENSITIVE,
   ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(query);
rs.absolute(3);
rs.deleteRow();
```

주의: JDBC구동프로그람에 따라 삭제방법이 다르다는데 대해 주의해야 한다. 어떤 구동프로그람들에서는 지워진 행이 결과모임에 보이지 않도록 완전히 없애버리며 또 일부 구동프로그람들에서는 삭제된 행을 빈행으로 남겨둔다.

3.6.3. 결과모임에서 변화보기

ResultSet에서 이루어진 변화들이 결과모임 그 자체나 다른 거래에서 반드시 보이는것이 아니다*.

다음의것들을 비롯하여 변화들의 가시성에 영향을 줄수 있는 인자들은 여러가지이다.

- JDBC구동프로그람의 실현
- 실시중에 있는 거래격리준위
- 결과모임의 류형

응용프로그람에서는 다음과 같은 DatabaseMetaData메쏘드들을 호출하여 결과모임에서 만들어진 변화들이 결과모임 그 자체에 보이게 할것인가 말것인가를 결정할수 있다.

- ownUpdatesAreVisible(int ResultSet.TYPE_XXX)
- ownDeletesAreVisible(int ResultSet.TYPE_XXX)
- ownInsertAreVisible(int ResultSet.TYPE_XXX)

DatabaseMetaData대면부는 또한 응용프로그람이 JDBC구동프로그람이 특정한 결과모임류형에 대한 변화들을 검출할수 있는가 없는가를 확인할수 있게 하는 다음과 같은 메쏘드들을 제공한다.

- insertsAreDetected(ResultSet.TYPE_XXX)
- deletesAreDetected(ResultSet.TYPE_XXX)
- updatesAreDetected(ResultSet.TYPE_XXX)

이 메쏘드들이 참을 돌려주는 경우 다음의 메쏘드들이 ResultSet에 대한 변화들을 검출하는데 리용될수 있다.

- wasInserted()
- wasDeleted()
- wasUpdated()

---

* 여기서 《보인다》, 《보이지 않는다》는것은 다음과 같은 의미를 가진다.

갱신이 보인다는것은 갱신이 이루어진 후 갱신된 값이 적당한 getter메쏘드의 호출에 의해 검색될수 있는 경우를 말한다. getter메쏘드가 여전히 초기의 렬값을 돌려준다면 갱신은 보이지 않는다고 말한다. 이와 류사하게 삽입된 행이 insertRow()호출이후 ResultSet에 나타나면 보이는것이다. 삭제는 삭제된 행이 결과모임에서 제거되거나 결과모임에 빈행을 남긴다면 보이는것이다.

자료가 변경된 다음 ResultSet를 닫고 그것을 다시 열면 변화된 내용이 항상 보인다는것을 명심하기 바란다.

가장 최근의 자료를 얻기 위한 다른 방도는 자료기지로부터 직접 어떤 행에 대하여 제일 마지막에 변경된 값을 주는 refreshRow()메쏘드를 리용하는것이다. 요구하는 행에 유표를 놓고 refreshRow()메쏘드를 호출하면 된다.

```
rs.absolute(3);
rs.refreshRow();
```

주의: 이때 결과모임은 TYPE_SCROLL_SENSITIVE여야 한다. rsType가 TYPE_SCROLL_INSENSITIVE인 ResultSet객체에 대하여 refreshRow()메쏘드는 아무런 작용도 하지 않는다.

3.6.4. 행모임

자료기지로부터 자료를 얻는 다른 방도는 RowSet객체를 리용하는것이다. RowSet는 결과모임 혹은 표형식의 어떤 다른 자료원천으로부터 얻은 행들의 모임을 포함하는 객체이다. RowSet는 ResultSet의 확장으로서 JDBC API에 JavaBeans지원을 추가한 보충기능을 가진다.

이와 류사하게 RowSetMetaData대면부는 ResultSetMetaData대면부를 확장한것이다.

RowSet는 JavaBeans이므로 속성들의 설정과 얻기 그리고 사건통지에서 JavaBeans 모형을 따른다. 따라서 프로그람에서 다른 구성요소들과의 결합이 쉽다.

RowSet는 표형식의 자료를 망으로 쉽게 보낼수 있게 해준다. 또한 밑에 놓인 JDBC 구동프로그람이 흘리기가능한 결과모임 혹은 갱신가능한 결과모임을 지원하지 않을 때 그것들을 제공하는 포장제로도 리용된다.

RowSet에는 접속형과 비접속형의 두가지가 있다.

- 접속형 RowSet는 ResultSet처럼 RowSet가 사용중에 있는 동안 자료원천에 대한 접속을 계속 유지한다.
- 비접속형 RowSet는 자료를 적재하거나 자료원천에 대한 변화를 반영할 때에만 자료원천에 대한 접속을 유지하고 대부분의 시간에는 접속을 해제한다.

접속이 해제되여있는 동안 RowSet는 JDBC구동프로그람이나 완전한 JDBC API를 요구하지 않는다. 비접속형 RowSet는 자료원천과의 접속을 계속 유지하지 않기때문에 자기의 자료를 기억기에 저장한다.

비접속형 RowSet는 자기가 포함하고있는 렬들에 대한 MetaData와 자기의 내부상태에 대한 정보를 보존하고있다. 또한 접속얻기와 지령실행, 자료원천에 대한 읽기와 쓰기를 위한 메쏘드들을 가지고있다.

RowSet의 실현들에는 다음과 같은것들이 포함된다.

- JDBCRowSet: 주로 JDBC구동프로그람이 만든 ResultSet객체를 감싸주는 포장제의 역할을 수행하는 접속형 RowSet
- CachedRowSet: 자기의 자료를 기억기에 캐쉬하는 비접속형 RowSet
- WebRowSet: 자료접근을 제공하는 자바써블레트와 통신하기 위하여 내부적으로 HTTP규약을 리용하는 접속형 RowSet

RowSet의 작성과 속성설정

RowSet는 JavaBeans이기때문에 속성들의 검색과 설정을 위한 설정자와 취득자메쏘드들을 가진다. 이 메쏘드들에는 다음과 같은것들이 포함된다.

- setCommand 실행되여야 할 SQL지령
- setConcurrency 읽기전용 혹은 갱신가능
- setType 흘리기가능 혹은 전진만 가능
- setDataSourceName DataSource접근에 리용
- setUrl DriverManager접근에 리용
- setUsername 자료기지에 접근하는 사용자이름 설정
- setPassword 자료기지에 접근하는 사용자통과암호 설정
- setTransactionIsolation 거래격리준위의 설정

사용자에게는 RowSet의 특정한 리용에 필요한 이 속성들의 설정만이 필요하다. 다음의 코드는 흘리기가능 및 갱신가능한 CachedRowSet객체 crset를 만든다.

```
CachedRowSet crset = new CachedRowSet();

crset.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
crset.setConcurrency(ResultSet.CONCUR_UPDATABLE);
crset.setCommand("SELECT * FROM Customers");
crset.setDataSourceName("jdbc/customers");
crset.setUsername("myName");
crset.setPassword("myPwd");
crset.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);

crset.addRowSetListener(listener);
```

접속을 만드는데 DriverManager를 리용하였다면 개발자는 JDBC URL, 사용자이름, 통과암호를 위한 속성들을 설정하여야 한다. 접속을 얻는데는 사용자이름 및 통과암호와 함께 DataSource객체를 리용하는것이 더 좋다.

CachedRowSet가 생성되고 초기화된 다음에는 execute()메쏘드만 호출하면 된다. 즉 RowSet는 자기의 속성정보들을 리용하여 접속을 얻고 질문을 실행한다. 그다음 RowSet에 있는 자료가 접근되고 갱신될수 있다.

RowSet의 사건

RowSetEvent는 렬값의 변화와 같이 RowSet에서 무엇인가 중요한 사건이 일어날 때 발생한다. RowSet가 JavaBeans라는데로부터 RowSet가 변화될 때 감시자(listener)에게 통지하는데 자바사건모형을 리용할수 있다.

아래에 RowSetListener메쏘드들을 렬거한다.

- rowChanged – RowSet가 변화될 때 호출된다
- rowSetChanged – RowSet가 삽입, 갱신 혹은 삭제가 진행될 때 호출된다
- cursorMoved – RowSet의 유표가 이동될 때 호출된다

제7절. 메타자료

메타자료(MetaData)는 JDBC API를 리용하여 얻을수 있는 자료기지나 그것의 내용에 대한 정보이다. JDBC에서 접근할수 있는 메타자료의 기본류형들은 다음과 같다.

- DatabaseMetaData
- ResultSetMetaData
- ParameterMetaData

3.7.1. 자료기지메타자료

DatabaseMetaData대면부는 자료기지 전체에 대한 정보를 제공한다. 이 대면부는 자료기지에 대한 다음과 같은 류형의 정보를 제공하는 150개이상의 메쏘드들을 정의한다.

- 자료원천에 대한 일반정보
- 자료원천에 대한 제한
- 거래지원의 준위
- 지원특징
- 원천이 포함하는 SQL객체들에 대한 정보

대부분의 DatabaseMetaData메쏘드들은 ResultSet속에 정보를 돌려줌으로서 사용자들이 이러한 정보를 검색하는데 getString과 getInt와 같은 ResultSet메쏘드들을 리용할수 있게 한다. MetaData의 주어진 형식을 얻을수 없는 경우 이 메쏘드들은 SQLException를 발생시킨다.

DatabaseMetaData의 일부 메쏘드들은 SQL문자렬들에 대한 표준적인 통용문자규칙에 맞는 문자렬패턴인수들을 가진다. 만일 탐색패턴인수가 null로 설정되면 그 인수의 검색조건은 탐색에서 무시된다.

만일 구동프로그람이 MetaData메쏘드를 지원하지 않는다면 표준적으로 SQLException가 발생한다. ResultSet를 돌려주는 메쏘드들의 경우 ResultSet(비여있을수도 있다.)가 돌려지든가 혹은 SQLException가 발생한다.

DatabaseMetaData객체는 Connection.getMetaData()메쏘드로 생성된다. 다음 그것은 자료기지에서 표의 이름들을 얻는 다음의 실례에서처럼 자료기지에 대한 정보를 얻는데 리용된다.

```
Connection con = DriverManager.getConnection("jdbc:odbc:Customers");
DatabaseMetaData dbmd = con.getMetaData();
ResultSet rs = dbmd.getTables(null,null,"%",new String[]{"TABLE"});
```

자료기지에 대한 일반정보는 다음과 같은 메쏘드들을 리용하여 접근할수 있다.

- getURL()
- getUserName()
- getDatabaseProductName()
- getSQLKeywords()
- nullsAreSortedHigh()와 nullsAreSortedLow()

지원된 기능에 대한 정보검색에 리용되는 메쏘드들은 다음과 같다.

- supportsBatchUpdates()
- supportsStoredProcedures()
- supportsFullOuterJoins()
- supportsPositionedDelete()

다음의 메쏘드들은 자료기지에 부과된 제한들을 결정하기 위하여 제공된다.

- getMaxRowSize()
- getMaxStatementLength()
- getMaxConnections()
- getMaxColumnsInTable()

SQL객체들과 그의 속성들에 대한 정보검색에 쓸모있는 메쏘드들에는 다음의 것들이 속한다.

- getSchemas()
- getCatalogs()
- getTables()
- getPrimaryKeys()
- getProcedures()

자료기지관리체계의 거래지원능력들은 다음의 메쏘드들을 리용하여 질문될수 있다.

- supportsMultipleTransactions()
- getDefaultTransactionIsolation()
- supportsSavePoints()

주의: 대부분의 DatabaseMetaData메쏘드들은 JDBC 2.0과 JDBC 3.0에서 추가 및 변경되였다. 때문에 사용자의 구동프로그람이 JDBC 2.0나 JDBC 3.0에 맞지 않는다면 SQLException가 발생할수 있다.

3.7.2. ResultSetMetaData

ResultSet에서 렬들에 대한 정보는 그의 getMetaData()메쏘드를 호출하여 리용할수 있다. 이 메쏘드에서 귀환된 ResultSetMetaData객체는 그 ResultSet객체의 렬수와 자료형, 속성들을 준다.

ResultSetMetaData접근에 리용할수 있는 일부 메쏘드들은 다음과 같다.

메쏘드	기능
getColumnCount()	ResultSet의 렬개수를 돌려준다.
getColumnDisplaySize(int column)	렬의 표준최대폭을 돌려준다.
getColumnLabel(int column)	인쇄와 현시에 리용할 렬제목을 돌려준다.
getColumnName(int column)	렬이름을 돌려준다.
getColumnType(int column)	렬의 SQL자료형색인을 돌려준다.
getColumnTypeName(int column)	렬의 SQL자료형이름을 돌려준다.
getPrecision(int column)	렬에서 10진수들의 수를 돌려준다.
getScale(int column)	소수점 오른쪽에 있는 수자들의 자리수를 돌

메쏘드	기능
	려준다.
getTableName(int column)	표이름을 돌려준다.
isAutoIncrement(int column)	렬에 자동적으로 번호가 붙을 때 참을 돌려준다.
isCurrency(int column)	렬값들이 화페형인 경우 참을 돌려준다.
isNullable(int column)	렬값이 NULL로 설정될수 있는 경우 참을 돌려준다.

목록 3-10은 렬이름들과 렬개수를 모르는 경우 ResultSetMetaData의 메쏘드들인 getColumnCount와 getColumnLabel를 리용하는 방법을 보여준다.

목록 3-10. ResultSetMetaData의 리용

```
public void PrintResultSet(String query){
  try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection("jdbc:odbc:Inventory");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    ResultSetMetaData md = rs.getMetaData();

    int nColumns = md.getColumnCount();
    for (int i=1;i<=nColumns;i++) {
System.out.print(md.getColumnLabel(i)+((i==nColumns)?"\n":"\t"));
    }
    while (rs.next()) {
      for(int i=1;i<=nColumns;i++){
        System.out.print(rs.getString(i)+((i==nColumns)?"\n":"\t") );
      }
    }
  }
  catch(ClassNotFoundException e){
    e.printStackTrace();
  }
  catch(SQLException e) {
    e.printStackTrace();
  }
}
```

3.7.3. ParameterMetaData

PreparedStatement의 getMetaData()메쏘드는 PreparedStatement가 실행되였을 때 돌려질 렬들에 대한 설명을 포함하는 ResultSetMetaData객체를 생성한다. 아래에 그 실례를 보여준다.

```
PreparedStatement ps = con.PrepareStatement("SELECT * FROM CUSTOMERS");
ResultSetMetaData md = ps.getMetaData();
int cols = md.getColumnCount();
```

getParameterMetaData()메쏘드는 PreparedStatement가 리용하는 IN 및 OUT파라메터들에 대한 설명을 포함하는 ParameterMetaData객체를 돌려준다.

```
PreparedStatement ps = con.PrepareStatement("SELECT * FROM CUSTOMERS");
ParameterMetaData pd = ps.getParameterMetaData();
int pType = pd.getParameterType(1);
```

주의: ParameterMetaData에 대한 지원은 JDBC 3.0 API에서 실현되였으며 JDK 1.4를 요구한다.

제8절. JDBC자료형

3.8.1. SQL자료형과 JDBC자료형과의 대응

JDBC 핵심부 API에서는 SQL자료형과 자바자료형사이의 자동적인 형변환을 제공한다. 표 3-5에 이 변환들을 보여주었다.

표 3-5. SQL형에 대한 자바자료형의 표준대응

SQL type	Java Type	Description
CHAR	String	고정길이 문자렬. 길이 n의 CHAR형에 대하여 자료기지관리체계는 저장 혹은 미사용공간의 채우기에 고정적으로 n개 문자를 할당한다.
VARCHAR	String	가변길이 문자렬. 길이 n의 VARCHAR형에 대하여 자료기지관리체계는 요구되는 만큼 n문자까지의 저장을 할당한다.
LONGVARCHAR	String	가변길이 문자렬. JDBC는 Java입력흐름처럼 LONGVARCHAR의 검색을 가능하게 한다.
NUMERIC	java.math.BigDecimal	임의의 정확도를 가진 부호붙은 10진수. BigDecimal나 String을 리용하여 검색할수 있다.
DECIMAL	java.math.BigDecimal	임의의 정확도를 가진 부호붙은 10진수. BigDecimal나 String을 리용하여 검색할수 있다.
BIT	boolean	예 /아니 값
TINYINT	byte	8 bit 옹근수값
SMALLINT	short	16 bit 옹근수값
INTEGER	int	32 bit 옹근수값
BIGINT	long	64 bit 옹근수값
REAL	float	단정확도 류점수
FLOAT	double	배정확도 류점수

SQL type	Java Type	Description
DOUBLE	double	배정확도 류점수
BINARY	byte[]	byte배렬처럼 검색
VARBINARY	byte[]	byte배렬처럼 검색
LONGVARBINARY	byte[]	바이트배렬처럼 검색. JDBC는 Java입력흐름으로서 LONGVARCHAR를 검색할수 있게 한다.
DATE	java.sql.Date	java.util.Date를 둘러싸는 얇은 포장
TIME	java.sql.Time	java.util.Date를 둘러싸는 얇은 포장
TIMESTAMP	java.sql.Timestamp	java.util.Date와 분리된 나노초(ns)값의 합성

일부 자료기지들은 어떤 렬들에 자동적으로 생성된 열쇠값들을 할당한다. 이 경우 삽입문은 그 렬에 값을 공급하는 책임을 지지 않으며 자료기지가 유일한 값을 생성하여 삽입한다. 이것은 보통 유일한 기본열쇠를 생성하는데 리용된다. 이 방법과 관련한 문제는 삽입이 진행된 후 그 값들을 얻는것이 어렵다는것이다. JDBC 3.0명세는 삽입후에 이 값들에 접근할수 있게 하는 보다 기능적인 Statement대면부를 정의하였다.

3개의 렬을 가진 USERS라는 표를 생각해보자. 《성》과 《이름》렬은 VARCHAR형이다. USER_ID렬은 자동생성되며 표에서 매 사용자에 대한 유일한 식별자를 포함한다. 아래에 실례를 보여준다.

```
Statement stmt = conn.createStatement();
String SQLInsert = "INSERT INTO Users (Family_Name,
   Personal_Name) "+ "VALUES('리', '수정')";

stmt.executeUpdate(SQLInsert);
ResultSet rs = stmt.getGeneratedKeys();
```

3.8.2. SQL3 자료형

JDBC 2.0 확장 API는 일반적으로 SQL3자료형이라고 불리우는 새로운 자료형을 지원한다. 이 새로운 자료형은 다음과 같은 특징들을 지원한다.

- 대단히 큰 자료객체
- 객체관계자료형

SQL3 자료형들은 ISO SQL표준의 다음 판본에 반영되여있다. JDBC API확장은 이 SQL3자료형들을 자바언어로 넘기기 위한 대면부들을 제공한다. 이 새로운 대면부들에 의해 사용자들은 SQL3 자료형들도 다른 자료형들과 같은 방법으로 리용할수 있다.

1) 객체관계형자료기지

객체관계형자료기지는 자료기지세계에 대한 객체지향설계의 리용을 지원하는 관계형자료기지관리체계의 확장이다.

례를 들어 보통의 관계형자료기지관리체계에서 다음의 렬들을 포함하는 이름과 주소에 대한 표를 작성할수 있다.

Family_Name	VARCHAR(20)
Personal_Name	VARCHAR(20)
Sex	CHAR(2)
State	VARCHAR(30)
City	VARCHAR(30)
District	VARCHAR(30)
Dong	VARCHAR(30)

또 다른 프로그람에서는 아마 마당크기도 다르고 보충적인 마당들을 더 가지고있는 이름과 주소에 관한 다른 표를 만들수 있다. 자료기지설계의 관점에서 보면 일률적으로 리용할수 있는 클라스나 표구조를 정의하는 능력은 대단히 매력적이다.

객체관계형자료기지는 사용자정의자료형(User Defined Data Types: UDT)과 더불어 이 방법을 지원하는 도구들을 제공한다.

2) SQL3 자료형의 리용

JDBC 2.0 확장이 제공하는 새로운 SQL3자료형은 다음과 같은것들을 지원한다.

- 대량의 자료를 바이트값 그대로 저장할수 있는 BLOB형
- 대량의 문자자료를 저장할수 있는 CLOB형
- 배렬을 렬값으로 저장할수 있는 ARRAY형
- 사용자정의형
- 구조화된 객체관계형
- DISTINCT형

다음의 목록에서는 SQL3형과 대응하는 JDBC 2.0대면부들을 보여준다.

- Blob구체레는 SQL BLOB값과 대응된다.
- Clob구체레는 SQL CLOB값과 대응된다.
- Array구체레는 SQL ARRAY값과 대응된다.
- Struct구체레는 SQL 구조형값과 대응된다.
- Ref구체레는 SQL REF값과 대응된다.

SQL자료형들은 표 3-6에서 보여준 메쏘드들을 리용하여 다른 자료형들과 같은 방식으로 검색, 저장 및 갱신된다.

표 3-6. SQL3 자료형 참조메쏘드들

SQL3자료형	얻기	설정	갱신
BLOB	getBlob	seBlob	updateBlob
CLOB	getClob	setClob	updateClob
ARRAY	getArray	setArray	updateArray
Structured type	getObject	setObject	updateObject
REF(structured type)	getObject	setObject	updateObject

이 새로운 자료형에 접근하는 한가지 실례를 아래에 보여준다. 이 코드는 환자의 진단레코드로부터 CLOB값인 Notes를 검색한다.

```
ResultSet rs = stmt.executeQuery(
   "SELECT Notes FROM Patients WHERE SSN = '123-45-6789'");
rs.next();
Clob notes = rs.getClob("Notes");
```

SQL의 BLOB, CLOB 혹은 ARRAY객체는 대단히 크기때문에 이 형들의 임의의 구체레는 실제로 SQL 위치자(locator) 혹은 그 구체레가 나타내는 자료기지의 객체에 대한 론리적인 지적자이다. JDBC는 자료기지봉사기에서 사용자의 의뢰기까지 자료를 전부 날라가지 않고도 그것들을 다룰수 있는 도구들을 제공한다. 이러한 특징을 리용하면 프로그람의 성능을 크게 개선할수 있다.

의뢰기에 BLOB나 CLOB값자료를 날라오려는 경우 Blob와 Clob대면부에 제공되여있는 다음과 같은 메쏘드들을 리용할수 있다.

getAsciiStream()	Clob객체에 의해 지적된 CLOB값을 ASCII 바이트들의 흐름으로 얻는다.
getCharacterStream()	Clob내용들을 Unicode흐름으로 얻는다.
getSubString(long pos, int length)	Clob객체에 의해 지적된 CLOB값에 있는 부분문자렬의 복사를 돌려준다.
length()	이 Clob객체에 의해 지적된 CLOB값에서 문자들의 수를 돌려준다.
Position(Clob searchstr, long start)	지적된 Clob객체 searchstr가 이 Clob객체에 나타나는 문자위치를 결정한다.
Position(String searchstr, long start)	지적된 부분문자렬 searchstr가 CLOB에 나타나는 문자위치를 결정한다.

Blob와 Clob객체는 둘다 의뢰기에서 객체의 값을 구체화하고 객체의 길이를 얻으며 객체의 값내에서 탐색을 수행하기 위한 메쏘드들을 제공한다.

JDBC 3.0 API확장에서는 BLOB와 CLOB의 값들을 직접 바꾸기 위한 메쏘드들을 추가한다.

- Blob.setBytes()
- Clob.setString()

JDBC Array객체는 결과모임이나 자바의 배렬처럼 역할하는 SQL ARRAY을 구체화한다. 례를 들어 다음의 코드는 Meds렬의 SQL ARRAY값을 java.sql.Array객체로 추출한 다음 의뢰기에 ARRAY값을 구체화한다.

```
ResultSet rs = stmt.executeQuery(
   "SELECT MEDS FROM Patients WHERE SSN = 123-45-6789");
while (rs.next()) {
   Array Medications = rs.getArray("MEDS");
   String[] meds = (String[])Medications.getArray();
   for (int i=0; i<meds.length; i++) {
      ...
   }
}
```

ResultSet의 getArray메쏘드는 아래에 보여준것처럼 현재 행의 MEDS마당에 저장된 값을 java.sql.Array객체 Medications에 돌려준다.

```
Array Medications = rs.getArray("MEDS");
```

변수 Medications는 봉사기의 SQL ARRAY에 대한 론리지적자를 의미하는 위치자를 포함하고있다. 그것은 ARRAY 그 자체의 요소를 포함하지는 않는다.

다음 행에서 getArray는 Array.getArray메쏘드로서 변수 meds에 할당되기전에 String객체들의 배렬로 형변환되는 자바객체를 돌려준다.

```
String[] meds =(String[])Medications.getArray();
```

결국 Array.getArray메쏘드는 의뢰기의 SQL ARRAY요소들을 String객체들의 배렬로 구체화한다.

3) 사용자정의자료형의 작성

SQL에서는 CREATE TYPE문으로 사용자정의자료형 즉 UDT를 작성할수 있다. 사용자가 정의할수 있는 자료형에는 두가지가 있다.

- 구조화된 자료형
- DISTINCT형

구조화된 자료형의 작성

다음의 SQL문은 새로운 자료형 ADDRESS를 작성하고 하나의 자료형으로 자료기지에 등록한다.

```
CREATE TYPE ADDRESS
(
   STATE VARCHAR(8),
   CITY VARCHAR(10),
   DISTRICT VARCHAR(12),
   DONG VARCHAR(14),
   NEIGHBOR INTEGER
);
```

이 정의에서 새로운 자료형 ADDRESS는 자바클라스의 마당들과 동등하게 볼수 있는 5가지 속성들을 가진다.

DISTINCT형의 작성

DISTINCT형은 한개의 속성만을 가진 구조화된 형으로 생각할수 있다. DISTINCT형은 항상 이미 정의되여있는 다른 자료형에 기초하고있다. 그러나 다른 UDT에 기초하지는 않는다는데 주의해야 한다. DISTINCT형들은 의거하는 형들에 적합한 메쏘드들을 리용하여 검색되거나 설정될수 있다.

례를 들어 결코 산수연산에 리용하려는것이 아닌 특수한 처리를 위한 좋은 후보로 될수 있는 주민등록번호(Social Security Number : SSN)형을 작성할수 있다. 아래에 그 실례를 보여준다.

```
CREATE TYPE SSN AS CHAR(9);
```

이것은 다음의 SQL Server지령과 동등하다.

```
EXEC sp_addtype SSN, 'VARCHAR(9)'
```

사용자정의자료형도 새로운 UDT를 정의하는데 리용될수 있다. 아래에 우에서 정의한 사용자정의자료형 ADDRESS와 SSN을 리용하여 새로운 UDT EMPLOYEE를 정의하는 실례를 보여주었다.

```
CREATE TYPE EMPLOYEE
(
   Emp_ID INTEGER,
   Family_Name VARCHAR(10),
   Personal_Name VARCHAR(10),
   Residence ADDRESS,
   Social SSN
);
```

이 정의는 접속을 열고 Statement를 만드는 보통의 방법으로 JDBC응용프로그람에서 작성될수 있다. 다음 실례는 구조화된 자료형 EMPLOYEE 의 정의를 자료기지에 보내기 위한 코드이다.

```
String createEmployee = "CREATE TYPE EMPLOYEE ("+
                        "Emp_ID INTEGER," +
                        "Family_Name VARCHAR(10)," +
                        "Personal_Name VARCHAR(10)," +
                        "Residence ADDRESS," +
                        "Social SSN);";
stmt.executeUpdate(createEmployee);
```

경우에 따라 코드에서 오유가 발생할수도 있다. 자바는 SQLExceptions을 던지므로써 이 오유들을 처리할수 있다.

제9절. 례외와 기록

자료기지에 접근할 때 여러가지 형태의 례외가 나타날수 있다. 가장 일반적인 례외는 SQLException이다.

3.9.1. SQLException

SQLException클라스는 java.lang.Exception을 확장한것이며 자료기지접근오유들에 대한 정보를 제공한다. 매 SQLException은 다음과 같은 정보를 제공한다.

- getMessage()메쏘드를 리용하여 얻을수 있는 자바례외통보문
- getSQLState()메쏘드를 리용하여 얻을수 있는 XOPEN SQL문 약속에 준한 SQL문 문자렬
- getErrorCode()메쏘드를 리용하여 얻을수 있는 제작업체에 따르는 옹근수오유코드. 표준적으로는 이것이 자료기지가 돌려주는 실제적인 오유코드이다.

이 외에 SQLException는 추가적인 오유정보를 제공하는 다음의 례외들을 얻을수 있게 한다.

SQLWarning

SQLWarning클라스는 SQLException을 확장한것으로서 자료기지접근경고들에 대한 정보를 제공한다. 발생하는 경고들은 경고를 일으키는 메쏘드를 가진 객체에 련결되여있으며 그 클라스의 getWarnnig()메쏘드로 귀환될수 있다.

SQLWarning은 SQLException으로부터 계승한 메쏘드외에도 추가적인 정보를 얻기 위하여 다음 SQLWarning을 얻거나 사슬에 경고를 추가하는 메쏘드들을 제공하고있다.

BatchUpdateException

BatchUpdateException은 일괄갱신기간에 일어나는 문제들에 대한 정보를 제공한다. BatchUpdateException은 SQLException을 확장한것이며 executeBatch메쏘드에 의하여 귀환되는 배렬과 류사한 갱신개수의 배렬을 더 가지고있다. 사용자는 다음과 같이 getUpdateCounts()메쏘드를 리용하여 이 배렬을 검색할수 있다.

```
Int[] updateCounts = b.getUpdateCounts();
```

갱신개수가 지령들과 같은 순서이기때문에 사용자는 묶음에서 어느 지령들이 성공적으로 실행되였는가를 알수 있다.

3.9.2. 기록

가장 간단한 응용프로그람들을 제외한 대부분의 응용프로그람들에서는 일부 오유등급과 사건기록을 결합할 필요가 있다. 물론 기록의 가장 기초적인 형식은 례외와 중요한 사건들에 대하여 통보하는 System.err와 System.out이다.

실천에서 단순히 체계조종탁에 례외통보문을 내보내는것은 일반적으로 적절하지 못하다. 사건기록과 오유기록들을 관리하기 위해서는 전용의 기록파일을 리용하는것이 좋다.

오유정보와 사건기록들을 파일에 쓰기 위해서는 간단히 System.err를 내보내거나 StackTrace를 찍어내기 위한 PrintWriter를 Exception클라스에서 리용할수 있다.

목록 3-11은 오유기록파일에 례외들을 기록하기 위한 다음의 두가지 방법을 실례들기 위하여 목록 3-1의 실례를 확장한것이다.

- printStackTrace()메쏘드를 리용하기 위한 PrintWriter를 정의한다.
- System.setErr()를 리용하여 System.err를 기록파일에 출력한다.

목록 3-11. 파일에 오유기록

```
package JavaDB_Bible.ch03.sec04;

import java.io.*;
import java.sql.*;
import java.util.*;

public class Logging {
   public static void main(String args[]) {
      PrintWriter errLog = null;
      PrintStream stderr = null;
      try {
         FileOutputStream errors = new FileOutputStream(
               "StdErr.txt", true);
         stderr = new PrintStream(errors);
         errLog = new PrintWriter(errors, true);
      } catch (Exception e) {
         System.out.println("Redirection error: " +
                     "Unable to open SystemErr.txt");
      }
      System.setErr(stderr);

      int qty;
      float cost;
      String name;
      String desc;
      String query =
```

```
                "SELECT Name, Description, Qty, Cost, Sell_Price FROM Stock";
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
                    "jdbc:odbc:Inventory");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                name = rs.getString("Name");
                desc = rs.getString("Description");
                qty = rs.getInt("Qty");
                cost = rs.getFloat("Cost");
                System.out.println(name + ", " + desc + "\t: " +
                            qty + "\t@ $" + cost);
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace(errLog);
        } catch (SQLException e) {
            System.err.println((new GregorianCalendar()).getTime());
            System.err.println("Thread:" + Thread.currentThread());
            System.err.println("ErrorCode:" + e.getErrorCode());
            System.err.println("SQLState:" + e.getSQLState());
            System.err.println("Message:" + e.getMessage());
            System.err.println("NextException: " + e.getNextException());
            e.printStackTrace(errLog);
            System.err.println();
        }
        try {
            stderr.close();
        } catch (Exception e) {
            System.out.println("Redirection error:" +
                        "Unable to close SystemErr.txt");
        }
    }
}
```

실례에서 현재시간과 현재스레드를 기록되는 오유정보의 부분으로 보관하는것은 실천에서 아무런 의의도 없다.

```
String query = "SELECT Name, Description, Qty, Cost, Sell_Price FROM
Stock";
```

우에서 서술한 query문자렬은 존재하지 않는 렬을 선택하려 하기때문에 SQL례외가 발

생하고 다음과 같은 오유통보문들이 StdErr.txt파일에 기록된다.

```
Tue Apr 02 14:38:46 GMT+09:00 2007
Thread:Thread[main,5,main]
ErrorCode:207
SQLState:S0022
Message:[Microsoft][ODBC SQL Server Driver][SQL Server]Invalid column
name 'Sell_Price'.
NextException: null
java.sql.SQLException: [Microsoft][ODBC SQL Server Driver][SQL Server]Invalid
column name 'Sell_Price'.
at sun.jdbc.odbc.JdbcOdbc.createSQLException(JdbcOdbc.java:6879)
at sun.jdbc.odbc.JdbcOdbc.standardError(JdbcOdbc.java:7036)
at sun.jdbc.odbc.JdbcOdbc.SQLExecDirect(JdbcOdbc.java:3065)
at sun.jdbc.odbc.JdbcOdbcStatement.execute(JdbcOdbcStatement.java:338)
at
sun.jdbc.odbc.JdbcOdbcStatement.executeQuery(JdbcOdbcStatement.java:253)
at Logging.main(Logging.java:31)
```

3장에 대한 요약

이 장에서는 JDBC API의 리용에 대하여 개괄하였다. 이 장에서 독자들은 JDBC에 기초한 응용프로그람을 이루는 기본적인 객체들에 대하여 학습하였다.

- DriverManager와 각이한 류형의 JDBC구동프로그람들의 리용
- 공용 및 분산접속들에 대한 JDBC DataSource들의 리용
- 접속리용
- Statement, PreparedStatement, CallableStatement의 리용
- 거래, 격리준위와 보관점의 리용
- 일괄갱신의 취급
- ResultSet와 Rowset의 리용
- MetaData의 리용
- SQL자료형을 JDBC로 넘기기
- 례외와 오유기록

제 4 장. 2 층모형에서 JDBC 와 SQL 의 리용

이 장에서는 이미 학습한 JDBC핵심 API와 SQL개념들을 2층구성방식의 응용프로그람실례를 통하여 더욱 공고히 한다.

제1절. JDBC와 SQL에 의한 표작성

4.1.1. JDBC를 리용한 표작성

표를 만들기에 앞서 먼저 자료기지를 창조한다. 자료기지의 창조는 자료기지관리체계자체에 의해 진행된다. GUI를 지원하는 MS Access, SQL Sever, Sybase와 Oracle과 같은 자료기지관리체계에서는 도형적인 수단을 리용한 자료기지작성방법을 제공한다. MySQL과 같이 지령행입력방식을 리용하는 자료기지관리체계에서는 자료기지관리체계를 실행시키고 입력재촉행에 다음과 같이 입력한다.

```
CREATE DATABASE 자료기지이름;
```

이 장의 실례들에서는 JDBC-ODBC다리를 리용하지만 임의의 JDBC구동프로그람들에 다 적용할수 있다. 사용자가 다른 구동프로그람을 리용하는 경우에는 DriverManager를 리용하여 구동프로그람을 등록할 때 자기가 리용하는 구동프로그람이름을 지적하면 된다. 여기서는 JDBC-ODBC다리를 리용하기로 하였으므로 창조된 자료기지를 ODBC Data Source Administrator프로그람을 리용하여 새롭게 등록해야 한다.

자료기지를 창조하였으므로 이제는 표를 만들수 있다. 여기서는 표 4-1과 같은 거래자정보를 관리하는 CONTACT_INFO표를 JDBC로 작성하는 실례를 보기로 한다.

표 4-1. 표에 대한 실례

Contact_ID	Family_Name	Personal_Name	Sex	State	City	District	Dong	Neighbor
100	김	성철	남	평양시	평양	중구역	교구동	32
101	김	은희	녀	평양시	평양	보통강구역	대보동	17

JDBC API는 해당 자료기지에 알맞는 구동프로그람의 적재, 자료기지에로의 접속, SQL지령의 생성과 실행 그리고 임의의 귀환되는 레코드들에 대한 처리 등 일련의 과제들

을 조종하는 몇개의 중요한 클라스들과 대면부들로 이루어져있다. 실례에서 우리가 사용해야 할 기본적인 클라스들은 다음과 같다.

- DriverManager
- Driver
- Connection
- Statement

DriverManager는 JDBC구동프로그람들을 적재하며 적당한 구동프로그람에 대한 접속을 돌려주는 책임을 진다.

우리가 처음으로 할것은 Class.forName()을 리용하여 이름에 의해 구동프로그람(우리의 경우 sun.jdbc.odbc.JdbcOdbcDriver)을 적재하는것이다. 다음 아래의 지령을 사용하여 DriverManager에 그것을 등록한다.

```
DriverManager.registerDriver(new JdbcOdbcDriver());
```

다음에는 아래의 지령을 리용하여 DriverManager와 자료기지의 접속을 요구한다.

```
getConnection(jdbc:driverName:databaseName);
```

DriverManager클라스는 지적된 URL에 대한 접속을 창조할수 있는 첫 구동프로그람을 찾기 위하여 등록된 모든 구동프로그람들을 조사한다. getConnection()메쏘드는 URL과 함께 자료기지에 접속하는 사용자이름과 통과암호를 주거나 자바속성객체를 넘길수 있게 한다.

```
getConnection(String url, String user, String password);
getConnection(String url, Properties info);
```

접속은 지정된 자료기지와의 대화접속(session)을 표현하며 SQL문들이 실행되고 결과들이 귀환될수 있는 토대로 된다.

Statement는 SQL질문 그 자체라기보다 접속을 통하여 자료기지에 SQL질문을 넘겨주는 Java클라스이다. Statement객체는 정적인 SQL문을 실행하고 그 결과를 얻어내는데 리용된다.

사용자가 자료기지에 보내는 실제 SQL지령은 우리가 CREATE지령을 론의할 때 작성했던 지령이다. JDBC는 구동프로그람에 그 어떤 질문렬도 다 넘겨주기때문에 응용프로그

람들이 비록 어떤 자료기지관리체계들에서 오유통보를 받을수 있게 되더라도 원하는만큼 충분히 SQL기능을 리용할수 있다. 목록 4-1은 JDBC를 리용하여 표를 작성하는 코드를 포함하고있다.

목록 4-1. JDBC를 리용한 표의 작성

```
package JavaDB_Bible.ch04.sec01;

import java.sql.*;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class TableMaker{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Contact_Info";
   static String url = "jdbc:odbc:";
   static String SQLCreate =
      "CREATE TABLE Contact_Info("+
      "Contact_ID INTEGER NOT NULL PRIMARY KEY, "+
      "Family_Name VARCHAR(10) NOT NULL, "+
      "Personal_Name VARCHAR(10) NOT NULL, "+
      "Sex VARCHAR(2) NOT NULL, "+
      "State VARCHAR(10) NOT NULL, "+
      "City VARCHAR(10) NULL, "+
      "District VARCHAR(10) NULL, "+
      "Dong VARCHAR(10) NOT NULL, "+
      "Neighbor VARCHAR(10) NOT NULL);";

   public TableMaker(){
      registerDriver();
   }

   public void setDatabaseName(String dbName){
      this.dbName = dbName;
   }

   public void registerDriver(){
      try{
         Class.forName(jdbcDriver);
      }catch(ClassNotFoundException e){
         System.err.println(e.getMessage());
      }
   }
```

```
    public void execute(String SQLCommand){
        url += dbName;
        Connection con;
        Statement stmt;
        try{
            con = DriverManager.getConnection(url);
            stmt = con.createStatement();
            stmt.execute(SQLCommand);
            con.close();
            if (con != null) con.close();
            if (stmt != null) stmt.close();
        }
        catch(SQLException e){
            System.err.println(e.getMessage());
        }
    }

    public static void main(String[] args){
        TableMaker tableMaker = new TableMaker();
        tableMaker.execute(SQLCreate);
    }
}
```

우의 실례를 콤파일하고 실행시키면 Contacts자료기지에 새로운 표 Contact_Info가 생긴다. MySQL과 같은 지령행방식의 자료기지관리체계를 리용할 때는 입력재촉행에 다음과 같이 입력해야 한다.

```
SHOW TABLES;
```

4.1.2. 표변경

표를 작성하는것과 함께 이미 있는 표를 변경할 필요가 있을수 있다. 실례로 표작성이 끝난 다음 추가적으로 전화번호와 전자우편주소에 대한 마당을 포함시키려고 한다고 하자. 많은 자료기지관리체계에서는 ALTER TABLE지령을 리용하여 표를 수정할수 있다. ALTER TABLE지령을 가지고 할수 있는 조작은 두가지이다.

- 이미 있는 표에 새로운 렬을 추가
- 이미 존재하는 렬을 변경

ALTER TABLE지령의 문장구성법은 다음과 같다.

```
ALTER TABLE 표이름 ADD 렬이름 자료형;
```

다음 실례는 CONTACT_INFO표에 전화번호마당을 추가하고있다.

```
ALTER TABLE CONTACT_INFO ADD PHONE VARCHAR(20);
```

렬의 제약조건을 NOT NULL로부터 NULL로 변경시키기 위해서는 MODIFY를 리용할수 있다.

```
ALTER TABLE 표이름 MODIFY 렬이름 자료형 NULL;
```

이런 방법으로 렬의 크기를 변화시키려면 다음과 같이 한다.

```
ALTER TABLE 표이름 MODIFY 렬이름 자료형;
```

경고: 렬의 크기를 항상 변화시킬수 있지만 그 크기는 렬에 들어갈수 있는 가장 큰 값 보다 작아서는 안된다. 류사하게 렬에 NULL값들이 없는 경우 렬의 제약조건을 NOT NULL로부터 NULL로만 변경할수 있다. MODIFY문은 자료기지관리체계에 따라 지원하지 않을수도 있다.

JDBC를 리용하여 표를 작성할수 있은것처럼 JDBC를 리용하여 표를 변경할수 있다. 목록 4-2에서 볼수 있는것처럼 표를 변경하는 코드는 새롭게 추가된 execute(String[] SQLcommand)메쏘드를 제외하면 TableMaker.java실례프로그람과 매우 류사하다. 이 메쏘드는 ALTER TABLE지령을 각각 실행하기 위해 SQL지령들의 배렬을 처음부터 끝까지 순환한다.

목록 4-2. JDBC를 리용한 표의 변경

```
package JavaDB_Bible.ch04.sec01;

import java.sql.*;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class TableModifier{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Contacts_Info";
   static String url = "jdbc:odbc:Contacts";

   static String[] SQLAlter = {
      "ALTER TABLE Contact_Info ADD Phone VARCHAR(16);",
      "ALTER TABLE Contact_Info ADD Email VARCHAR(50);"
   };
```

```
  public TableModifier(){
    registerDriver();
  }

  public void registerDriver(){
    try{
      Class.forName(jdbcDriver);
      DriverManager.registerDriver(new JdbcOdbcDriver());
    }
    catch(ClassNotFoundException e){
      System.err.print(e.getMessage());
    }
    catch(SQLException e){
      System.err.println(e.getMessage());
    }
  }

  public void execute(String[] SQLCommand){
    try{
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      for(int i=0;i<SQLCommand.length;i++){
        stmt.execute(SQLCommand[i]);
      }
      con.close();
    }
    catch(SQLException e){
      System.err.println(e.getMessage());
    }
  }

  public static void main(String[] args){
    TableModifier tableModifier = new TableModifier();
    tableModifier.execute(SQLAlter);
  }
}
```

SQL을 리용한 표의 삭제는 DROP TABLE지령으로 수행한다. DROP TABLE지령은 표와 관련된 뷰와 색인들도 함께 삭제한다.

DROP TABLE지령의 문장구성법은 다음과 같다.

```
DROP TABLE 표이름;
```

CONTACT_INFO표를 삭제하기 위한 지령은 다음과 같다.

```
DROP TABLE Contact_Info;
```

표를 삭제하는 코드는 표를 작성하거나 변경하는 코드와 매우 류사하기때문에 해당한 Java실례프로그람을 제시하지 않았다. 뒤에서 취급되는 TableBuilder프로그람(목록 4-3)에 표를 삭제하기 위한 실례프로그람코드가 포함되여있다.

4.1.3. Swing에 기초한 표구축자작성

이제부터 Swing을 기초로 하는 표구축자를 만들어보자. 이 프로그람은 조금만 손질하면 임의의 자료기지관리체계에서 동작할수 있는 완벽한 자료기지관리도구이다.

표구축자는 MVC(Model View Controller)구성방식을 리용하고있다. MVC설계는 리해하기 쉽고 만들기도 쉬우며 유지보수도 쉽다.

첫 단계는 MVC구성방식의 조종자(controller)부분을 작성하는것이다. 조종자는 다양한 창문요소들로부터의 사용자입력에 응답하며 사용자의 지령들을 실행하기 위한 모델을 조종한다. 사용자입력은 JMenu클라스와 대화칸들 그리고 JInternalFrame클라스들로부터 들어온다. 일어나는 사건들은 다음과 같다.

1. 사용자가 자료기지를 선택한다.
2. 사용자가 표이름을 할당한다.
3. 사용자가 표에서 필요한 마당들을 정의한다.
4. 만들려는 표를 생성하기 위해 CREATE TABLE지령이 발행된다.

조종자는 목록 4-1에서 본 코드에 기초한 클라스들을 리용하는 모델과 호상작용한다. MVC구성방식의 현시부분은 자료기지이름이나 표이름과 같이 단일한 값입력들만을 취급하는 JOptionPane, 임의의 보다 복잡한것들을 다루기 위한 전용의 JInternalFrame들과 함께 보통의 JMenu항목들에 의해 조종된다. 창문요소들은 초기화기간과 수집한 자료를 돌려주기 위해서만 조종자와 호상작용한다.

MVC구성방식의 모형부분은 자료기지에 접속하고 CREATE TABLE지령을 발행하는 실제적인 JDBC기능들을 수행한다.

조종자

조종자는 JFrame에 기초하고있으며 JFrame 역시 창문요소들을 조종할수 있다. 구축자는 JFrame을 구축하는외에 JInternalFrame클라스와 JMenu클라스 그리고 JMenu의 ActionListener를 다루기 위한 JDesktopPane을 추가한다.

사용자가 [Database]차림표항목을 선택하였을 때 selectDatabase()메쏘드가 호출

된다. 이 메쏘드는 JOptionPane을 리용하여 사용자에게 자료기지이름을 문의한다. 자료기지이름이 보관된 다음 [New Table]차림표항목과 [Drop Table]차림표항목이 능동으로 된다.

[New Table]과 [Drop Table]차림표항목중에 어느 하나가 선택되면 표이름을 얻기 위하여 JOptionPane이 현시되고 선택한 차림표에 따라 사용자가 표를 작성할수 있게 해주는 TableBuilderFrame이 현시되든지 표를 삭제하겠는가를 확인하는 JOptionFrame이 현시된다.

사용자가 표이름에 대한 재촉문에 응답하면 displayTableBuilderFrame()메쏘드가 호출된다. 이 메쏘드는 JInternalFrame으로부터 완전한 CREATE TABLE지령을 받기 위하여 ActionListener, CommandListener를 설정하는 TableBuilderFrame을 일으킨다.

마지막에 CommandListener가 자료기지에 접속하여 표를 작성하는 JDBC의 SQLToolkit클라스에 CREATE TABLE지령을 넘겨보낸다.(목록 4-3)

목록 4-3. Swing기초의 표구축자 - 기본 JFrame

```
package JavaDB_Bible.ch04.sec01;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class DBManager extends JFrame{
   JMenuBar menuBar = new JMenuBar();
   JDesktopPane desktop = new JDesktopPane();
   String database = null;
   String tableName = null;
   String menuSelection = null;
   TableBuilderFrame tableMaker = null;
   DatabaseUtilities dbUtils = null;

   TableMenu tableMenu = new TableMenu();

   MenuListener menuListener = new MenuListener();

   public DBManager(){
      setJMenuBar(menuBar);
      setTitle("JDBC Database Bible");
```

```
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(desktop, BorderLayout.CENTER);
        setSize(new Dimension(550, 400));

        menuBar.add(tableMenu);
        tableMenu.setMenuListener(menuListener);

        setVisible(true);
    }

    private void displayTableBuilderFrame(){
        tableName = JOptionPane.showInputDialog(this, "Table:",
            "Select table", JOptionPane.QUESTION_MESSAGE);
        tableMaker = new TableBuilderFrame(tableName);
        tableMaker.setCommandListener(new CommandListener());
        desktop.add(tableMaker);
        tableMaker.setVisible(true);
    }

    private void selectDatabase(){
        database = JOptionPane.showInputDialog(this, "Database:",
            "Select database", JOptionPane.QUESTION_MESSAGE);
        dbUtils = new DatabaseUtilities();
        dbUtils.setDatabaseName(database);
        dbUtils.setExceptionListener(new ExceptionListener());

        tableMenu.enableMenuItem("New Table", true);
        tableMenu.enableMenuItem("Drop Table", true);
    }

    private void executeSQLCommand(String SQLCommand){
        dbUtils.execute(SQLCommand);
    }

    private void dropTable(){
        tableName = JOptionPane.showInputDialog(this,"Table:",
            "Select table",JOptionPane.QUESTION_MESSAGE);
        int option = JOptionPane.showConfirmDialog(null,
            "Dropping table "+tableName,
            "Database "+database,
            JOptionPane.OK_CANCEL_OPTION);
        if(option==0){
```

```
            executeSQLCommand("DROP TABLE "+tableName);
        }
    }

    class MenuListener implements ActionListener{
        public void actionPerformed(ActionEvent event){
            String menuSelection = event.getActionCommand();
            if(menuSelection.equals("Database")){
                selectDatabase();
            }else if(menuSelection.equals("New Table")){
                displayTableBuilderFrame();
            }else if(menuSelection.equals("Drop Table")){
                dropTable();
            }else if(menuSelection.equals("Exit")){
                System.exit(0);
            }
        }
    }

    class ExceptionListener implements ActionListener{
        public void actionPerformed(ActionEvent event){
            String exception = event.getActionCommand();
            JOptionPane.showMessageDialog(null,exception,
                "SQL Error",JOptionPane.ERROR_MESSAGE);
        }
    }

    class CommandListener implements ActionListener{
        public void actionPerformed(ActionEvent event){
            String SQLCommand = event.getActionCommand();
            executeSQLCommand(SQLCommand);
        }
    }

    public static void main(String args[]){
        DBManager dbm = new DBManager();
    }
}
```

창문

창문은 기본적으로 두개의 클라스에 의해서 처리된다.

- TableMenu
- TableBuilderFrame

TableMenu는 자료기지를 선택하고 표를 식별하는데 리용되는 기초적인 JMenu로부터 받은 입력들을 현시하고 처리한다. TableMenu는 JMenuBar의 첫번째 차림표이므로 역시 Exit기능을 처리한다.

TableMenu는 일반적인 기능들을 제공하는 기초적인 DBMenu(목록 4-4)를 확장하고 있다. DBMenu의 기본목적은 차림표작성을 간단하게 하고 차림표항목들이 조종자로부터 개별적으로 설정되지 않도록 차림표항목들에 사건감시자를 붙이기 위한 공통점을 제공하는것이다.

목록 4-4. DBMenu(TableMenu의 기초클라스)

```
package JavaDB_Bible.ch04.sec01;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DBMenu extends JMenu{
   JMenuItem dbItem;
   JMenuItem newItem;
   JMenuItem openItem;
   JMenuItem exitItem;
   ActionListener menuListener = null;
   MenuItemListener itemListener = new MenuItemListener();

   public DBMenu(){
   }

   public void enableMenuItem(String itemName, boolean enable){
      Component c[] = getMenuComponents();
      for(int i=0;i<c.length;i++){
         if(c[i] instanceof JMenuItem){
            JMenuItem menuItem = (JMenuItem)c[i];
            If (menuItem.getText().equals(itemName))
menuItem.setEnabled(enable);
```

```
            }
        }
    }

    public void setMenuListener(ActionListener menuLlistener){
        this.menuListener = menuLlistener;
    }

    class MenuItemListener implements ActionListener{
        public void actionPerformed(ActionEvent event){
            String action = event.getActionCommand();
            if(action != null) menuListener.actionPerformed(event);
        }
    }
}
```

DBMenu클라스는 JMenuItem들의 생성을 위해 단순한 기초클라스를 제공하는 JMenuItem클라스를 확장한 DBMenuItem클라스에 의해 지원된다.

목록 4-5. DBMenuItem(JMenuItem을 작성하기 위한 편의클라스)

```
package JavaDB_Bible.ch04.sec01;

import java.awt.event.*;
import javax.swing.*;

public class DBMenuItem extends JMenuItem {
    public DBMenuItem(String name, char hotkey,
            ActionListener itemListener, boolean enabled) {
        super(name,(int)hotkey);
        setActionCommand(name);
        setEnabled(enabled);
        addActionListener(itemListener);
    }
}
```

이러한 편의클라스들을 리용하면 우리가 만들려는 차림표들의 작성은 목록 4-6에서 보는것처럼 매우 간단해진다.

목록 4-6. TableMenu클라스

```
package JavaDB_Bible.ch04.sec01;

import javax.swing.*;

public class TableMenu extends DBMenu {
   JMenuItem dbItem;
   JMenuItem newItem;
   JMenuItem openItem;
   JMenuItem exitItem;

   public TableMenu() {
      setText("Table");
      setActionCommand("Table");
      setMnemonic((int)'T');

      dbItem = new DBMenuItem("Database", 'D', itemListener, true);
      newItem = new DBMenuItem("New Table", 'T', itemListener, false);
      openItem = new DBMenuItem("Drop Table", 'D', itemListener, false);
      exitItem = new DBMenuItem("Exit", 'X', itemListener, true);

      add(dbItem);
      addSeparator();
      add(newItem);
      add(openItem);
      addSeparator();
      add(exitItem);
   }
}
```

TableBuilderFrame은 MVC의 창문에서 핵심이라고 말할수 있다. TableBuilderFrame은 JInternalFrame의 확장클라스로서 자료기지표에 마당들을 설정하는데 리용되는 JTable, 생성된 SQL지령의 미리보기를 제공하는 JTextArea, 그리고 조종자에 ActionEvent를 착화시키고 거기에 발생된 SQL지령을 보내는 <Create Table>단추를 포함하고있다.

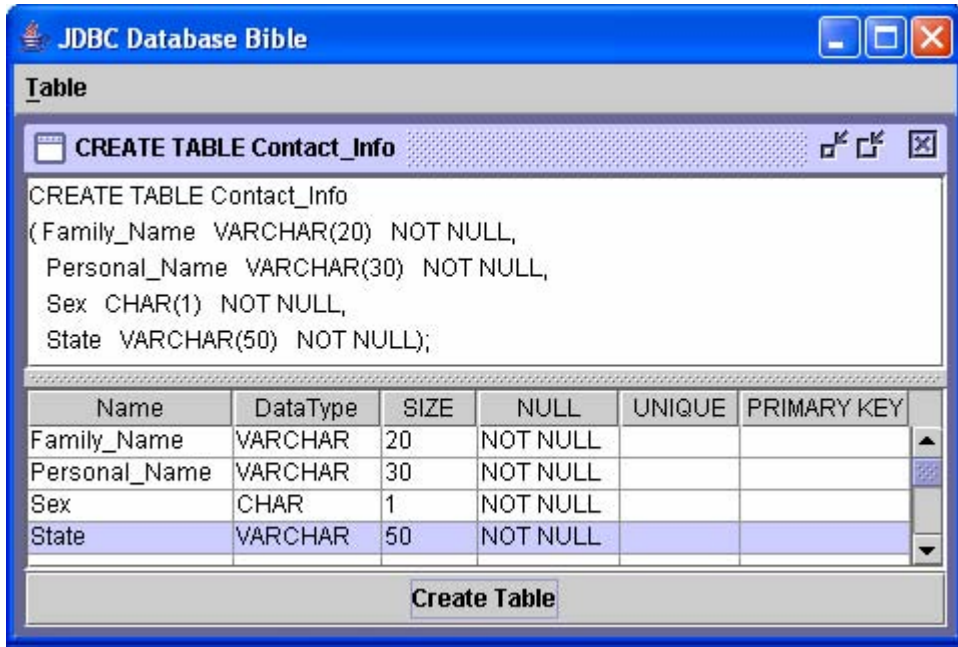



그림 4-1. TableBuilderFrame이 표기입항목들로부터 SQL을 생성

TableBuilderFrame은 JTable에 기초하여 구성되는데 JTable은 DataType와 같이 마당들에 대한 렬편집기로서 JComboBox구성요소들을 추가하여 주문된다.

프레임의 아래에 있는 <Create Table>단추를 누르면 TableBuilderFrame이 생성된 SQL지령을 조종자에 넘겨줄수 있도록 MVC조종자에 의해 setCommandListener() 메쏘드가 호출된다.

목록 4-7. TableBuilderFrame

```
package JavaDB_Bible.ch04.sec01;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class TableBuilderFrame extends JInternalFrame{
   protected int nRows = 15;
   protected int nColumns = 6;
   protected JTable table;
   protected JTextArea SQLPane = new JTextArea();
   protected JButton createButton = new JButton("Create Table");
```

```
    protected ActionListener commandListener = null;

    protected String tableName = null;
    protected String SQLCommand = "";
    protected String SQLCommandRoot = "";

    public TableBuilderFrame(String tableName){
        setSize(500, 320);
        setLocation(10,10);
        setClosable(true);
        setMaximizable(true);
        setIconifiable(true);
        setResizable(true);
        this.getContentPane().setLayout(new BorderLayout());
        this.tableName = tableName;
        SQLCommandRoot = "CREATE TABLE "+tableName;
        setTitle(SQLCommandRoot);
        init();
        setVisible(true);
    }

    // JInternalFrame의 초기화
    private void init(){
        table = createTable(nRows);
        TableChangeListener modelListener = new TableChangeListener();
        table.getModel().addTableModelListener(modelListener);
        JScrollPane sqlScroller = new JScrollPane(SQLPane);
        JScrollPane tableScroller = new JScrollPane(table);
        JSplitPane splitter = new

JSplitPane(JSplitPane.VERTICAL_SPLIT,sqlScroller,tableScroller);
        splitter.setDividerLocation(100);
        getContentPane().add(splitter,BorderLayout.CENTER);
        getContentPane().add(createButton,BorderLayout.SOUTH);
        createButton.addActionListener(new ButtonListener());
    }

    private JTable createTable(int nRows){
        String[] dataTypes = {"CHAR", "VARCHAR", "INT", "FLOAT", "DATE"};
        String[] defNull = {"", "NULL", "NOT NULL"};
        String[] defUnique = {"", "UNIQUE"};
        String[] defPriKey = {"", "PRIMARY KEY"};
```

```
        String[] colNames = {"Name", "DataType", "Size", "NULL", "UNIQUE",
"PRIMARY KEY"};
        String[][] rowData = new String[nRows][colNames.length];

        for(int i=0;i<nRows;i++){
            for(int j=0;j<colNames.length;j++)
                rowData[i][j]= "";
        }

        JComboBox dTypes = new JComboBox(dataTypes);
        JComboBox nullDefs = new JComboBox(defNull);
        JComboBox uniqueDefs = new JComboBox(defUnique);
        JComboBox primaryKDefs = new JComboBox(defPriKey);
        JTable table = new JTable(rowData,colNames);

        table.getColumnModel().getColumn(1).setCellEditor(new
DefaultCellEditor(dTypes));
        table.getColumnModel().getColumn(3).setCellEditor(new
DefaultCellEditor(nullDefs));
        table.getColumnModel().getColumn(4).setCellEditor(new
DefaultCellEditor(uniqueDefs));
        table.getColumnModel().getColumn(5).setCellEditor(new
DefaultCellEditor(primaryKDefs));
        return table;
    }

    public String parseTable(){
        String tableValues = "";
        int rows = table.getRowCount();
        int cols = table.getColumnCount();

        if(rows >=0 && cols >=0){
            tableValues += "\n( ";
            for(int i=0;i<rows;i++){
                String rowData = "";
                for(int j=0;j<cols;j++){
                    String field = (String)table.getValueAt(i,j);
                    if(field != null){
                        if(field.length()==0)break;
                        if(j==2)rowData += "(";
                        else if(i>0||j>0)rowData += "   ";
                        rowData += field;
```

```
                    if(j==2)rowData += ")";
                }
            }
            if(rowData.length()==0)break;
            tableValues += rowData + ",\n";
        }
    }

    if(tableValues.endsWith(",\n")){
        int tvLen = tableValues.length()-2;
        if(tvLen>0)    tableValues = tableValues.substring(0,tvLen);
    }
    tableValues += ");";
    return tableValues;
}

// CommandListener는 MVC조종자에 의하여 SQL지령을
// 되돌리기 위한 호출로 설정된다
public void setCommandListener(ActionListener commandListener){
    this.commandListener=commandListener;
}

// CreateButton에 대한 감시자
class ButtonListener implements ActionListener{
    public void actionPerformed(ActionEvent event){
        String action = event.getActionCommand();
        if(commandListener != null){
            ActionEvent evt = new ActionEvent(this,0,SQLCommand);
            commandListener.actionPerformed(evt);
        }
    }
}

// JTable우에서 일어나는 편집사건들에 대한 감시자
class TableChangeListener implements TableModelListener{
    public TableChangeListener(){
    }
    public void tableChanged(TableModelEvent event){
        SQLCommand = SQLCommandRoot+parseTable();
        SQLPane.setText(SQLCommand);
    }
}
}
```

Model

MVC모델의 모델부분은 이 장의 앞에서 우리가 만든 JDBC클라스에 지나지 않는다.

이 판본은 매몰된 SQL지령문자렬들과 그것을 시험하기 위하여 우리가 리용하는 main()메쏘드를 제거하여 가볍게 편집되였다.

또한 조종자에 의해 등록된 ExceptionListener에 ActionEvent를 착화시키는 례외처리를 변경하였다. ExceptionListener는 조종탁에 례외들을 인쇄하는것이 아니라 SQLToolkit로부터 례외들을 현시하기 위하여 JOptionPane이 튀여나오게 한다.

목록 4-8. DatabaseUtilities - JDBC코드

```
package JavaDB_Bible.ch04.sec01;

import java.awt.event.*;
import java.sql.*;
import java.util.Vector;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class DatabaseUtilities{
    static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
    static String dbName = "Inventory";
    static String urlRoot = "jdbc:odbc:";
    private ActionListener exceptionListener = null;

    public DatabaseUtilities(){
        registerDriver();
    }

    public void setDatabaseName(String dbName){
        this.dbName = dbName;
    }

    public void registerDriver(){
        try {
            Class.forName(jdbcDriver);
            DriverManager.registerDriver(new JdbcOdbcDriver());
        }
        catch(ClassNotFoundException e){
            reportException(e.getMessage());
        }
        catch(SQLException e){
            reportException(e.getMessage());
```

```
        }
    }

    public void execute(String SQLCommand){
        String url = urlRoot+dbName;
        try {
            Connection con = DriverManager.getConnection(url);
            Statement stmt = con.createStatement();
            stmt.execute(SQLCommand);
            con.close();
        }
        catch(SQLException e){
            reportException(e.getMessage());
        }
    }

    public void setExceptionListener(ActionListener exceptionListener) {
        this.exceptionListener = exceptionListener;
    }

    private void reportException(String exception) {
        if(exceptionListener!=null){
            ActionEvent evt = new ActionEvent(this,0,exception);
            exceptionListener.actionPerformed(evt);
        }else{
            System.err.println(exception);
        }
    }
}
```

제2절. JDBC와 SQL을 리용한 자료의 삽입, 갱신, 삭제

일단 자료기지를 창조하고 표를 구성하였으면 자료들을 삽입하고 삭제하며 변경하는 방법을 아는것이 중요하다. SQL에서는 사용자가 자료기지의 자료를 다루는데 리용할수 있는 다음과 같은 3개의 문들을 제공한다.

- INSERT
- UPDATE
- DELETE

4.2.1. 자료의 삽입

INSERT문은 하나의 행에서 개별적인 마당들을 삽입하는데가 아니라 표에 행전체를 삽입하는데 리용된다. INSERT문의 가장 단순한 리용형식은 표에 자료를 한번에 한개 행씩 삽입하는것이다. 이 문은 다른 표들에서 선택된 여러개의 행들을 삽입하기 위해 SELECT문과 결합하여 리용할수도 한다.

목록 4-9에 JDBC와 INSERT문을 함께 리용하는데 필요한 코드를 보여주었다. 이 실례는 목록 4-1의 코드와 류사한데 JDBC를 리용하여 표를 작성하는 방법을 설명하고있다. 이것은 JDBC API가 자료기지관리체계에 임의의 SQL지령을 넘겨주는 수단을 어떻게 제공하는가 하는 설명을 보충해준다.

목록 4-9. JDBC와 INSERT의 리용

```
package JavaDB_Bible.ch04.sec02;

import java.awt.event.*;
import java.sql.*;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class DataInserter{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Contacts";
   static String urlRoot = "jdbc:odbc:";

   public DataInserter(){
      registerDriver();
   }

   public void setDatabaseName(String dbName){
      this.dbName = dbName;
   }
```

```
    public void registerDriver(){
        try {
            Class.forName(jdbcDriver);
            DriverManager.registerDriver(new JdbcOdbcDriver());
        }
        catch(ClassNotFoundException e){
            System.err.println(e.getMessage());
        }
        catch(SQLException e){
            System.err.println(e.getMessage());
        }
    }

    public void execute(String SQLCommand){
        String url = urlRoot+dbName;
        try {
            Connection con = DriverManager.getConnection(url);
            Statement stmt = con.createStatement();
            stmt.execute(SQLCommand);
            con.close();
        }
        catch(SQLException e) {
            System.err.println(e.getMessage());
        }
    }


    public static void main(String args[]){
        DataInserter inserter = new DataInserter();
        String SQLCommand =
            "INSERT INTO Contact_Info " +
            "(Family_Name, Personal_Name, Sex, State, "+
            "City, District, Dong, Neighbor)"+
            "VALUES "+
            "('김', '성희', '녀', '평양시', '평양', "+
            "'서성구역', '하신동', '23');";
        inserter.execute(SQLCommand);
    }
}
```

이 실례를 콤파일하고 실행시키면 Contact_Info표에 새로운 레코드가 생긴것을 볼수 있다. MySQL과 같은 지령행방식의 자료기지관리체계를 리용하는 경우에는 지령재촉

행에 다음과 같이 입력해야 한다.

```
SELECT * FROM Contact_Info;
```

목록 4-9의 실례에 설명된 INSERT문은 주로 표에 레코드들을 단번에 삽입하도록 되여있다. 그렇지만 어떤 표의 부분적인 자료를 다른 표에 복사해야 할 경우가 때때로 제기된다. 이런 때 한번에 한개의 레코드를 전송한다면 매 레코드가 한개 표에서 개별적으로 검색되여 다른 표에 삽입되여야 하기때문에 시간이 많이 걸린다.

SQL에서는 요구되는 레코드들에 대하여 자료기지에 질문하는 SELECT지령과 INSERT지령을 결합하여 리용하게 함으로써 이 문제를 해결한다. 이렇게 하면 모든 처리가 관계형자료기지관리체계내부에서 진행되며 레코드들을 검색하고 그것들을 외부에서 재삽입하는데 드는 시간을 줄일수 있다.

JDBC와 INSERT...SELECT의 리용도 쉽다. 목록 4-9의 main()부분을 목록 4-10의 짤막한 코드로 교체하고 다시 실행시킨다면 《김》가성을 가진 사람들의 Family_Name과 Personal_Name이 들어있는 Names표를 작성할수 있다.

목록 4-10. JDBC와 INSERT...SELECT의 리용

```
public static void main(String args[]){
   DataInserter inserter = new DataInserter();
   String SQLCommand = "INSERT INTO Names "+
      "SELECT Family_Name, Personal_Name "+
      "FROM Contact_Info "+
      "WHERE Family_Name = '김';";
   inserter.execute(SQLCommand);
}
```

일단 표에 자료들을 입력한 다음에는 주소나 상품의 가격과 같이 자료마당들의 변화를 반영하기 위해 그것을 갱신할 필요가 제기된다.

4.2.2. UPDATE문을 리용한 자료의 갱신

UPDATE지령은 행들의 모임에서 개별적인 렬들의 내용을 변경하는데 리용된다. UPDATE지령은 보통 갱신하려는 행들을 선택하는 WHERE문절과 함께 리용된다.

자료기지응용프로그람에서 자주 요구되는것은 레코드들의 갱신이다. 실례로 거래자가 집을 이사하였을 때에는 그의 주소를 변경해야 한다.

```
UPDATE Contact_Info
SET District = '평천구역', Dong = '봉남동'
WHERE Family_Name = '김' AND Personal_Name = '성희';
```

목록 4-11에 JDBC와 UPDATE지령의 리용실례를 보여주었다.

목록 4-11. JDBC와 UPDATE의 리용

```
package JavaDB_Bible.ch04.sec02;

import java.awt.event.*;
import java.sql.*;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class DataUpdater{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Contacts";
   static String urlRoot = "jdbc:odbc:";
   private ActionListener exceptionListener = null;

   public DataUpdater(){
      registerDriver();
   }

   public void setDatabaseName(String dbName){
      this.dbName=dbName;
   }

   public void registerDriver(){
      try {
         Class.forName(jdbcDriver);
         DriverManager.registerDriver(new JdbcOdbcDriver());
      }
      catch(ClassNotFoundException e){
         System.err.println(e.getMessage());
      }
      catch(SQLException e){
         System.err.println(e.getMessage());
      }
   }

   public void execute(String SQLCommand){
      String url = urlRoot+dbName;
      try {
         Connection con = DriverManager.getConnection(url);
         Statement stmt = con.createStatement();
         stmt.execute(SQLCommand);
```

```
        con.close();
      }
      catch(SQLException e){
        System.err.println(e.getMessage());
      }
    }

    public  static  void main(String  args[]){
      DataUpdater  inserter  =  new  DataUpdater();
      String SQLCommand = "UPDATE Contact_Info "+
        "SET District = '평천구역', Dong = '봉남동'" +
        "WHERE Family_Name = '김' AND Personal_Name = '성희';";
      inserter.execute(SQLCommand);
    }
}
```

SQL지령을 실행하는데 리용된 기초코드는 변경되지 않은 상태로 남아있다. 시험해보기 위해 실례를 콤파일하고 실행시키면 Contact_Info표에서 변경된 레코드를 볼수 있다.

4.2.3. DELETE문을 리용한 자료의 삭제

마지막 자료조종문은 DELETE인데 전체 레코드나 레코드들의 그룹을 삭제하는데 리용된다. 삭제하려는 레코드들을 지정하는데 WHERE문절을 리용한다.

DELETE문의 리용은 대단히 간단하다. 아래의 실례는 성이 《김》이고 이름이 《성희》인 레코드를 삭제하는데 리용하는 지령이다.

```
DELETE FROM Contact_Info
WHERE Family_Name = '김' AND Personal_Name = '성희';
```

주의: INSERT, DELETE와 UPDATE는 작업하고있는 표에서만이 아니라 다른 표들과 관련하여 참조의 완정성문제를 산생시킬수 있으므로 주의해야 한다.

JDBC와 DELETE지령의 리용은 SQL지령부분을 제외한다면 JDBC와 UPDATE지령의 리용실례와 같다.

4.2.4. JDBC에서 거래관리지령의 리용

거래는 순서대로 반드시 실행되여야만 하는 지령묶음 혹은 지령들의 련속적인 렬이다. 거래는 대표적으로 은행에서의 자금이동과 같이 여러개의 관련된 지령들을 포함한다. 만약 의뢰인 A의 구좌에서 의뢰인 B의 구좌에 자금을 전송하려고 한다면 적어도 두개의 자

료기지접근지령들이 실행되여야 한다.

- 의뢰인 A의 구좌가 계산자리왼쪽(차방)에 기입되여야 한다.
- 의뢰인 B의 구좌가 계산자리오른쪽(대방)에 기입되여야 한다.

이 지령들중에 하나가 실행되고 다른 하나가 실행되지 않으면 자금이 의뢰인 B의 구좌에 나타나지 않은채 의뢰인 A의 구좌에서 없어지거나 자금이 의뢰인 A의 구좌에서 회수됨이 없이 은행에 적자를 내면서 의뢰인 B의 구좌에 나타나게 된다. 해결책은 론리적으로 련관된 지령들을 단일한 거래로 위탁되는 묶음으로 결합하는것이다. 그러면 문제가 발생하는 경우 전체 거래가 취소되여 파괴적인 혼란을 피할수 있다.

거래관리에서 리용되는 기본지령들은 COMMIT와 ROLLBACK이다. 대부분의 자료기지들은 모든 지령들이 실행될 때마다 개별적으로 관계형자료기지관리체계에 위탁을 지시하는 AUTOCOMMIT옵션을 지원한다. 거래과 함께 작업을 시작할 때에는 AUTOCOMMIT옵션을 OFF로 설정한 다음 거래에서 요구하는 모든 지령들이 완결된 다음에야 COMMIT지령을 실행한다. 거래과정에 어떤 문제가 발생하였다면 ROLLBACK지령을 리용하여 전체 거래를 취소할수 있다.

JDBC실례프로그람에서 거래관리지령의 리용은 아주 간단하다. 아래에 목록 4-11의 실례를 변경한 부분을 아래에 보여준다.

```
public void execute(String SQLCommand){
   String url = urlRoot + dbName;
   try{
      Connection con = DriverManager.getConnection(url);
      con.setAutoCommit(true);
      Statement stmt = con.createStatement();
      stmt.execute(SQLCommand);
      con.close();
   }
   catch(SQLException e){
      System.err.print(e.getMessage());
   }
}
```

setAutoCommit(true)행을 추가하면(5행) 자료기지관리체계에 모든 변화들에 대한 자동위탁이 위임된다. 변경된 코드를 콤파일하고 실행하면 원래의 실례를 실행시켰을 때와 같은 결과를 정확히 얻을것이다.

이번에는 아래의 코드와 같이 setAutoCommit(false)를 리용해보자.

```
public void execute(String SQLCommand){
   String url = urlRoot + dbName;
   try{
      Connection con = DriverManager.getConnection(url);
      con.setAutoCommit(false);
      Statement stmt = con.createStatement();
      stmt.execute(SQLCommand);
      con.close();
   }
   catch(SQLException e){
      System.err.print(e.getMessage());
   }
}
```

실례를 실행하면 "Invalid Transaction State"라는 례외가 던져지고 갱신이 진행되지 않는다. 접속닫기전에 거래를 끌내지 않으면 례외가 생긴다.

이제 try블로크에서 접속을 닫기전에 변화위탁을 지시하는 commit지령을 삽입해보자.

```
try{
    Connection con = DriverManager.getConnection(url);
    con.setAutoCommit(false);
    Statement stmt = con.createStatement();
    stmt.execute(SQLCommand);
    con.commit();
    con.close();
}
```

콤파일하여 실행하면 정확한 결과가 얻어진다.

만약 con.commit()를 con.rollback()로 교체한다면 변경이 취소되여 아무런 변화도 일어나지 않을것이다.

```
try{
    Connection con = DriverManager.getConnection(url);
    con.setAutoCommit(false);
    Statement stmt = con.createStatement();
    stmt.execute(SQLCommand);
    // con.commit();
    con.rollback();
    con.close();
}
```

갱신지령이 실행된 후 거래를 취소하기전에 갱신된 마당(여기에서는 District마당)의 갱신된 값을 읽어보기 위한 SELECT문을 삽입함으로써 UPDATE가 실행되였는지 안되였는지를 검사해볼수 있다.

```
try{
   Connection con = DriverManager.getConnection(url);
   con.setAutoCommit(false);
   Statement stmt = con.createStatement();
   stmt.execute(SQLCommand);

   String query = "SELECT District FROM Contact_Info "+
     "WHERE Family_Name = '김' AND Personal_Name = '성철';";
   ResultSet rs = stmt.executeQuery(query);
   rs.next();
   System.out.println("State = " + rs.getString(1));

   con.rollback();
   con.close();
}
```

이 코드를 실행시키면 화면에 District의 갱신된 값이 나타나지만 자료기지를 보면 변경이 취소되였기때문에 이전의 값이 여전히 남아있는것을 볼수 있다.

4.2.5. Swing에 의한 표편집기

이 소절에서 취급하는 문제들을 설명하기 위하여 앞에서 작성했던 Swing에 기초한 표구축자를 표편집기(Table Editor)로 확장하였다.(그림 4-2) 표편집기는 표구축자의 구성요소들로부터 파생된 요소들에 기초하고있다. 또한 새로운 [Edit]차림표(Insert, Update, Delete)와 **JInternalFrame**의 새로운 **JTable**(Insert, Edit, Delete기능을 처리하기 위한)이 추가되였다.

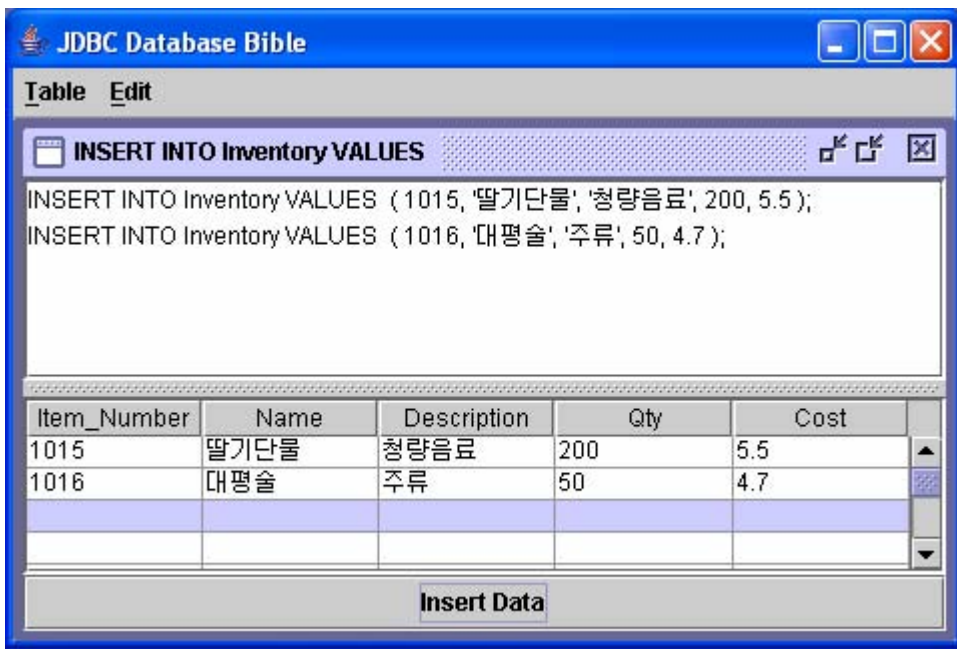



그림 4-2. INSERT지령으로 자료를 삽입

일어나는 사건들은 다음과 같다.

1. 사용자가 자료기지를 선택한다.
2. 사용자가 Insert, Update, Delete와 같은 동작을 선택한다.
3. 사용자가 표를 선택한다.
4. 사용자와 호상작용하기 위한 TableEdit틀이 현시된다.
5. SQL지령이 동적으로 작성되고 지령이 실행된다.

표편집기작성의 첫번째 단계는 DBMenu를 계승하여 [Edit]차림표를 작성하는것이다. DBMenuItem들인 Insert, Update, Delete가 [Edit]차림표에 추가되고 MainFrame클라스의 기초를 이루는 Jframe에 매달린다.

목록 4-12. Insert, Update와 Delete항목이 있는 [Edit]차림표

```
package JavaDB_Bible.ch04.sec02;

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
public class EditMenu extends DBMenu{
   JMenuItem insertItem;
   JMenuItem updateItem;
   JMenuItem deleteItem;

   public EditMenu(){
      setText("Edit");
      setActionCommand("Edit");
      setMnemonic((int)'E');

      insertItem=new DBMenuItem("Insert",'I',itemListener,false);
      updateItem=new DBMenuItem("Update",'U',itemListener,false);
      deleteItem=new DBMenuItem("Delete",'D',itemListener,false);

      add(insertItem);
      add(updateItem);
      add(deleteItem);
   }
}
```

앞에서 이미 언급한것처럼 DBMenu기초클라스와 DBMenuItem클라스는 차림표들을 구축하기 위한 편의클라스들이다. 이렇게 편의클라스들을 리용하면 차림표코드가 현저히 간소화된다.

TableEditFrame

목록 4-13에서 보여준 TableEditFrame은 앞에서 고찰한 TableBuilderFrame과 매우 류사하다. 그것은 JInternalFrame을 계승하고있으며 자료기지표에 해당한 마당들을 설정하는데 리용되는 JTable을 포함한다. 그것은 또한 JTextArea를 포함하는데 이 클라스는 발생된 SQL지령의 미리보기와 <Insert Data>단추를 제공한다.

목록 4-13. TableEditFrame

```
package JavaDB_Bible.ch04.sec02;

import java.awt.*;
import java.awt.event.*;
import java.util.EventObject;
import java.util.EventListener;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

class TableEditFrame extends JInternalFrame{
   protected JTable table;
   protected JTextArea SQLPane = new JTextArea();
   protected JButton insertButton = new JButton("Insert Data");
   protected DatabaseUtilities dbUtils = null;

   protected String tableName = null;
   protected String colNames[] = null;
   protected String dataTypes[] = null;

   protected String SQLCommand[] = null;
   protected String SQLCommandRoot = "";

   public TableEditFrame(String tableName, DatabaseUtilities dbUtils) {
      setSize(500, 320);
      setLocation(10, 10);
      setClosable(true);
```

```
        setMaximizable(true);
        setIconifiable(true);
        setResizable(true);
        getContentPane().setLayout(new BorderLayout());
        this.tableName = tableName;
        this.dbUtils = dbUtils;
        SQLCommandRoot = "INSERT INTO " + tableName + " VALUES ";
        setTitle(SQLCommandRoot);
        init();
        setVisible(true);
    }

    // JInternalFrame의 초기화
    private void init() {
        colNames = dbUtils.getColumnNames(tableName);
        dataTypes = dbUtils.getDataTypes(tableName);
        table = createTable(colNames, 15);
        TableChangeListener modelListener = new TableChangeListener();
        table.getModel().addTableModelListener(modelListener);

        JScrollPane sqlScroller = new JScrollPane(SQLPane);
        JScrollPane tableScroller = new JScrollPane(table);
        JSplitPane splitter = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                             sqlScroller, tableScroller);

        splitter.setDividerLocation(100);
        getContentPane().add(splitter, BorderLayout.CENTER);
        getContentPane().add(insertButton, BorderLayout.SOUTH);
        insertButton.addActionListener(new ButtonListener());
    }

    protected JTable createTable(String[] colNames, int nRows) {
        String[][] rowData = new String[nRows][colNames.length];
        for (int i=0; i<nRows; i++) {
            for (int j=0; j<colNames.length; j++) rowData[i][j] = "";
        }
        JTable table = new JTable(rowData, colNames);
        return table;
    }

    public Vector parseTable() {
        int rows = table.getRowCount();
```

```
    int cols = table.getColumnCount();
    Vector tableValues = new Vector();

    if (rows>=0 && cols>=0) {
        for (int i=0; i<rows; i++) {
            String rowData = "";
            for (int j=0; j<cols; j++) {
                String field = (String) table.getValueAt(i, j);
                if (field.length()>0) {
                    field = fixApostrophes(field);
                    if (j>0) rowData += ", ";
                    if (dataTypes[j].equalsIgnoreCase("char") ||
                        dataTypes[j].equalsIgnoreCase("varchar"))
                        rowData += "'" + field + "'";
                    else
                        rowData += field;
                }
            }
            if (rowData.length()==0)break;
            tableValues.addElement(" ( " + rowData + " );\n");
        }
    }
    return tableValues;
}

private String fixApostrophes(String in) {
    int n = 0;
    while ((n = in.indexOf("'", n)) >= 0) {
        in = in.substring(0, n) + "'" + in.substring(n);
        n += 2;
    }
    return in;
}

// InsertButton에 대한 감시자
class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        dbUtils.execute(SQLCommand);
    }
}

// JTable우에서 일어나는 편집사건에 대한 감시자
```

```
    class TableChangeListener implements TableModelListener {
        public TableChangeListener() {
        }

        public void tableChanged(TableModelEvent event) {
            Vector rowData = parseTable();
            SQLCommand = new String[rowData.size()];
            SQLPane.setText("");
            for (int i = 0; i < rowData.size(); i++) {
                if (rowData.elementAt(i) == null)break;
                SQLCommand[i] = SQLCommandRoot + (String) rowData.elementAt(i);
                SQLPane.append(SQLCommand[i]);
            }
        }
    }
}
```

parseTable()메쏘드는 TableBuilderFrame에서 보다 약간 변경되였으며 Vector 문자렬을 반환한다. 이렇게 변경하면 단추를 한번 눌러 여러개의 INSERT지령들을 발생시킬수 있다.

추가적인 변경은 TableChangeListener에 대하여 진행되였는데 사건체계를 통해서가 아니라 직접 DatabaseUtilities클라스에 접근한다. 이것은 한번의 단추찰칵사건에 대한 응답으로 여러개의 SQL지령들을 발행하기 위한 능력을 지원하기 위한것이다.

조종자클라스

목록 4-14에 조종자클라스인 DatabaseManager클라스를 보여주고있다. 이 클라스는 표구축자를 만들 때의 클라스에 기초하고있으며 새로운 JinternalFrame, TableEditFrame을 현시하기 위한 새로운 차림표와 새로운 메쏘드 displayTableEditFrame()에서 사건가로채기를 할수 있는 추가적인 코드를 통합하고있다. 추가된 부분들은 행의 마지막에 설명문을 추가하였다.

목록 4-14. DatabaseManager - Controller클라스

```
package JavaDB_Bible.ch04.sec02;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
```

```
public class DBManager extends JFrame{
   JMenuBar menuBar = new JMenuBar();
   JDesktopPane desktop = new JDesktopPane();
   String database = null;
   String tableName = null;
   String menuSelection = null;

   TableBuilderFrame tableMaker = null;
   TableEditFrame tableEditor = null;   // 추가된 부분
   DatabaseUtilities dbUtils = null;

   TableMenu tableMenu = new TableMenu();
   EditMenu editMenu = new EditMenu();   // 추가된 부분

   MenuListener menuListener = new MenuListener();

   public DBManager(){
      setJMenuBar(menuBar);
      setTitle("JDBC Database Bible");
      getContentPane().setLayout(new BorderLayout());
      getContentPane().add(desktop, BorderLayout.CENTER);
      setSize(new Dimension(540, 405));

      menuBar.add(tableMenu);
      tableMenu.setMenuListener(menuListener);

      menuBar.add(editMenu);    // 추가된 부분
      editMenu.setMenuListener(menuListener);

      setVisible(true);
   }

   private void displayTableBuilderFrame(){
      tableName = JOptionPane.showInputDialog(this, "Table:",
         "Select table", JOptionPane.QUESTION_MESSAGE);
      tableMaker = new TableBuilderFrame(tableName);
      tableMaker.setCommandListener(new CommandListener());
      desktop.add(tableMaker);
      tableMaker.setVisible(true);
   }
```

```
private void displayTableEditFrame(){   // 추가된 부분
   tableName = JOptionPane.showInputDialog(this,"Table:",
      "Select table",JOptionPane.QUESTION_MESSAGE);
   tableEditor = new TableEditFrame(tableName, dbUtils);
   desktop.add(tableEditor);
   tableEditor.setVisible(true);
}

private void selectDatabase(){
   database = JOptionPane.showInputDialog(this, "Database:",
      "Select database", JOptionPane.QUESTION_MESSAGE);
   dbUtils = new DatabaseUtilities();
   dbUtils.setDatabaseName(database);
   dbUtils.setExceptionListener(new ExceptionListener());

   tableMenu.enableMenuItem("New Table", true);
   tableMenu.enableMenuItem("Drop Table", true);

   editMenu.enableMenuItem("Insert", true);
   editMenu.enableMenuItem("Update", true);
   editMenu.enableMenuItem("Delete", true);
}

private void executeSQLCommand(String SQLCommand){
   dbUtils.execute(SQLCommand);
}

private void dropTable(){
   tableName = JOptionPane.showInputDialog(this,"Table:",
      "Select table",JOptionPane.QUESTION_MESSAGE);
   int option = JOptionPane.showConfirmDialog(null,
      "Dropping table "+tableName,
      "Database "+database,
      JOptionPane.OK_CANCEL_OPTION);
   if(option==0){
      executeSQLCommand("DROP TABLE "+tableName);
   }
}

class MenuListener implements ActionListener{
   public void actionPerformed(ActionEvent event){
      String menuSelection = event.getActionCommand();
```

```
        if(menuSelection.equals("Database")){
          selectDatabase();
        }else if(menuSelection.equals("New Table")){
          displayTableBuilderFrame();
        }else if(menuSelection.equals("Drop Table")){
          dropTable();
        }else if(menuSelection.equals("Insert")){
          displayTableEditFrame();
        }else if(menuSelection.equals("Exit")){
          System.exit(0);
        }
      }
    }

    class ExceptionListener implements ActionListener{
      public void actionPerformed(ActionEvent event){
        String exception = event.getActionCommand();
        JOptionPane.showMessageDialog(null,exception,
          "SQL Error", JOptionPane.ERROR_MESSAGE);
      }
    }

    class CommandListener implements ActionListener{
      public void actionPerformed(ActionEvent event){
        String SQLCommand = event.getActionCommand();
        executeSQLCommand(SQLCommand);
      }
    }

    public static void main(String args[]){
      DBManager dbm = new DBManager();
    }
}
```

JDBC에 의해 제공되는 가장 쓸모있는 도구들중의 하나는 ResultSet에서 결과자료에 대한 정보를 검색할수 있는 능력이다. 이 정보는 다음 소절에서 학습하는 JDBC ResultSetMetaData를 리용하여 얻을수 있다.

4.2.6. JDBC ResultSetMetaData

편집기에는 편집되고있는 표에 대한 정보를 얻기 위해 ResultSet MetaData클라스를 리용하는 두개의 메쏘드들이 추가되였다.

다음의 MetaData객체들은 표에 대한 필요한 정보를 돌려준다.

- 자료기지준위에서 정보를 돌려주는 DatabaseMetaData
- ResultSet준위에서 정보를 돌려주는 ResultSetMetaData

ResultSetMetaData객체를 리용하는 리유는 지금 현시되고있는 정보를 렬들에 대한 정보로 제한하기 위한것이며 ResultSet객체를 고찰할 때까지 DatabaseMetaData객체에 대한 론의를 미루기 위해서이다.

ResultSetMetaData는 렬이름과 자료형을 포함하여 표의 자료에 대한 여러가지 류형의 정보에 접근할수 있게 한다. 가장 쓸모있는 ResultSet MetaData의 메쏘드 몇가지를 아래에 보여준다.

- int getColumnCount()
- String getColumnName(int column)
- String getColumnTypeName(int column)

이 메쏘드들의 사용법은 매우 간단하다. 실례로 표에서 모든 렬들의 이름을 얻기 위해서는 먼저 ResultSetMetaData를 얻는데 리용되는 ResultSet를 돌려주는 간단한 질문을 실행한다.

```
String SQLCommand = "SELECT * FROM " + tableName + ";";

try {
   Connection con = DriverManager.getConnection(url);
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery(SQLCommand);
   ResultSetMetaData md = rs.getMetaData();

   columnNames = new String[md.getColumnCount()];
   for(int i=0;i<columnNames.length;i++){
      columnNames[i] = md.getColumnLabel(i+1);
   }
   con.close();
}
```

목록 4-15에서 보여준 DatabaseUtilities클라스의 확장판본에는 execute()메쏘드의 두번째 판본이 추가되였다. 이 새로운 판본은 여러개의 INSERT지령들을 순환할수 있도록 문자렬배렬인수를 받아들인다.

목록 4-15. DatabaseUtilities-JDBC코드

```
package JavaDB_Bible.ch04.sec02;

import java.awt.event.*;
import java.sql.*;
import java.util.Vector;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class DatabaseUtilities{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Inven";
   static String urlRoot = "jdbc:odbc:";
   private ActionListener exceptionListener = null;

   public DatabaseUtilities(){
      registerDriver();
   }

   public void setDatabaseName(String dbName){
      this.dbName = dbName;
   }

   public void registerDriver(){
      try {
         Class.forName(jdbcDriver);
         DriverManager.registerDriver(new JdbcOdbcDriver());
      }
      catch(ClassNotFoundException e){
         reportException(e.getMessage());
      }
      catch(SQLException e){
         reportException(e.getMessage());
      }
   }

   public void execute(String SQLCommand){
      String url = urlRoot+dbName;
```

```
    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      stmt.execute(SQLCommand);
      con.close();
    }
    catch(SQLException e){
      reportException(e.getMessage());
    }
  }

  public void execute(String[] SQLCommand){
    String url = urlRoot + dbName;
    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      for(int i=0;i<SQLCommand.length;i++){
        stmt.execute(SQLCommand[i]);
      }
      con.close();
    }
    catch(SQLException e){
      reportException(e.getMessage());
    }
  }

  public String[] getColumnNames(String tableName){
    Vector dataSet = new Vector();
    String[] columnNames = null;
    String url = urlRoot + dbName;
    String SQLCommand = "SELECT * FROM "+tableName+";";

    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(SQLCommand);
      ResultSetMetaData md = rs.getMetaData();

      columnNames = new String[md.getColumnCount()];
      for(int i=0;i<columnNames.length;i++){
        columnNames[i] = md.getColumnLabel(i+1);
      }
      con.close();
    }
```

```
    catch(SQLException e){
      reportException(e.getMessage());
    }
    return columnNames;
  }

  public String[] getDataTypes(String tableName){
    Vector dataSet = new Vector();
    String[] dataTypes = null;
    String url = urlRoot+dbName;
    String SQLCommand = "SELECT * FROM "+tableName+";";

    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(SQLCommand);
      ResultSetMetaData md = rs.getMetaData();

      dataTypes = new String[md.getColumnCount()];
      for(int i=0;i<dataTypes.length;i++){
        dataTypes[i] = md.getColumnTypeName(i+1);
      }
      con.close();
    }
    catch(SQLException e){
      reportException(e.getMessage());
    }
    return dataTypes;
  }

  public void setExceptionListener(ActionListener exceptionListener) {
    this.exceptionListener = exceptionListener;
  }

  private void reportException(String exception) {
    if(exceptionListener!=null){
      ActionEvent evt = new ActionEvent(this,0,exception);
      exceptionListener.actionPerformed(evt);
    }else{
      System.err.println(exception);
    }
  }
}
```

제3절. JDBC와 SQL질문을 리용한 자료검색

자료기지응용프로그람에서 가장 중요한 기능들중의 하나는 자료기지 표에서 레코드들을 찾고 요구하는 형식으로 그것들을 돌려주는것이다. 형식화된 레코드들의 탐색과 반환과정을 자료기지 **질문**(query)이라고 한다.

4.3.1. JDBC ResultSets

JDBC ResultSet는 질문에 의해 반환된 행과 렬들에 배렬된 자료를 유지한다. ResultSet는 자료의 현재행을 지적하는 유표를 유지한다. 유표는 next()메쏘드가 호출될 때마다 한개 행씩 아래로 이동한다. 자료는 행단위로 접근되며 필요한 자료는 ResultSet가 제공하는 취득자메쏘드에 렬이름이나 렬번호를 넘겨주어 얻어낸다.

- getXXX(String columnName)
- getXXX(int columnNumber)

취득자메쏘드는 렬자료를 SQL자료형으로부터 지적된 Java형으로 변환해준다. 렬이름을 사용하여 "getXXX"를 수행할 때 같은 이름을 가진 렬들이 여러개라면 첫번째로 만나는 렬값이 반환된다.

ResultSetMetaData

getMetaData()메쏘드에 의해 반환되는 ResultSetMetaData객체는 렬번호, 자료형, 렬의 속성과 같은 ResultSet의 렬에 대한 정보를 제공한다. 이미 3장에서 ResultSetMetaData객체를 고찰하였다.

아래에 ResultSetMetaData에 접근할수 있는 일부 메쏘드들을 렬거하였다.

- getColumnCount() - ResultSet에 있는 렬수를 돌려준다.
- getColumnLabel(int columnNumber) - 인쇄할 때나 혹은 화면에 현시할 때 리용되는 렬제목을 돌려준다.
- getColumnName(int columnNumber) - 렬이름을 돌려준다.
- getColumnTypeName(int columnNumber) - 렬의 SQL자료형이름을 돌려준다.

이 4개 메쏘드만 가지면 임의의 질문결과를 현시하는데 충분한 정보를 얻을수 있다.

JDBC로 결과모임을 돌려주기 위한 SELECT의 리용

자료기지로부터 자료를 검색하는 수속은 질문이기때문에 돌려지는 자료를 유지하기 위하여 ResultSet를 정의해야 한다는것을 제외하면 자료삽입에 리용된 수속과 대단히 류사하다. ResultSet뿐아니라 ResultSet에 대한 정보를 보관하고있는 ResultSetMetaData객체도 정의해야 한다. getData메쏘드는 실행되는 질문과 관련된 아무런 정보도 가지고있지 않기때문에 돌려지는 렬개수를 얻기 위해 이 객체를 리용한다.

목록 4-16의 실례는 단순히 ResultSet를 순환하면서 체계조종탁에 자료를 출력한다.

목록 4-16. JDBC를 리용한 자료검색

```
package JavaDB_Bible.ch04.sec03;

import java.awt.event.*;
import java.sql.*;
import java.util.Vector;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class DataRetriever{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Contacts";
   static String urlRoot = "jdbc:odbc:";
   private ActionListener exceptionListener = null;

   public DataRetriever(){
      registerDriver();
   }

   public void setDatabaseName(String dbName){
      this.dbName=dbName;
   }

   public void registerDriver(){
      try {
         Class.forName(jdbcDriver);
         DriverManager.registerDriver(new JdbcOdbcDriver());
      }
      catch(ClassNotFoundException e){
         reportException(e.getMessage());
      }
      catch(SQLException e){
         reportException(e.getMessage());
```

```
        }
    }

    public String[][] executeQuery(String SQLQuery){
        Vector dataSet = new Vector();
        String url = urlRoot + dbName;

        try {
            Connection con = DriverManager.getConnection(url);
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(SQLQuery);
            ResultSetMetaData md = rs.getMetaData();

            int nColumns = md.getColumnCount();
            while(rs.next()){
                String[] rowData = new String[nColumns];
                for(int i=0; i<nColumns; i++) {
                    rowData[i] = rs.getObject(i+1).toString();
                }
                dataSet.addElement(rowData);
            }
            con.close();
        }
        catch(SQLException e){
            reportException(e.getMessage());
        }

        String[][] records = new String[dataSet.size()][];
        for (int i=0;i<records.length;i++){
            records[i]=(String[])dataSet.elementAt(i);
        }
        return records;
    }

    public void setExceptionListener(ActionListener exceptionListener){
        this.exceptionListener=exceptionListener;
    }

    private void reportException(String exception){
        if(exceptionListener!=null){
            ActionEvent evt = new ActionEvent(this, 0, exception);
            exceptionListener.actionPerformed(evt);
```

```
    }else{
      System.err.println(exception);
    }
  }

  public static void main(String args[]){
    DataRetriever retriever = new DataRetriever();
    retriever.setDatabaseName("Contacts");
    String[][] records =
      retriever.executeQuery("SELECT * FROM Contact_info");
    for (int i=0;i<records.length;i++) {
      String[] record = records[i];
      for (int j=0;j<record.length;j++) {
        if (j>0) System.out.print("\t");
        System.out.print(record[j]);
      }
      System.out.println();
    }
  }
}
```

표에서 자료를 검색하기 위한 코드와 자료를 삽입하는데 리용되는 코드의 기본차이는 ResultSet와 ResultSetMetaData객체의 리용이라고 볼수 있다. 또 다른 차이는 ResultSet가 돌려질 때에는 execute()메쏘드가 아니라 Statement객체의 executeQuery()메쏘드를 리용해야 한다는것이다.

초기에 ResultSet의 유표는 첫 행전에 위치하고있으므로 첫번째 행으로 유표를 이동시키기 위해 ResultSet.next()메쏘드를 실행해야 한다.

목록 4-16에서는 또한 ResultSet에 있는 렬개수를 얻는데 ResultSetMetaData를 리용하고있다.

다음 체계에서는 질문들에 대한 실행능력을 추가하면서 1절과 2절에서 시작한 JDBC Swing실례프로그람을 계속 개발한다.

4.3.2. Swing에 의한 SQL질문판

질문판(그림 4-3)을 추가하여 Swing에 의한 표구축자를 확장해보자. 질문판은 1절에서 작성한 구성요소들에 기초하고있다. 질문판을 현시할수 있게 하는 새로운 [View]차림표와 질문을 다루기 위한 JInternalFrame을 추가하겠다.

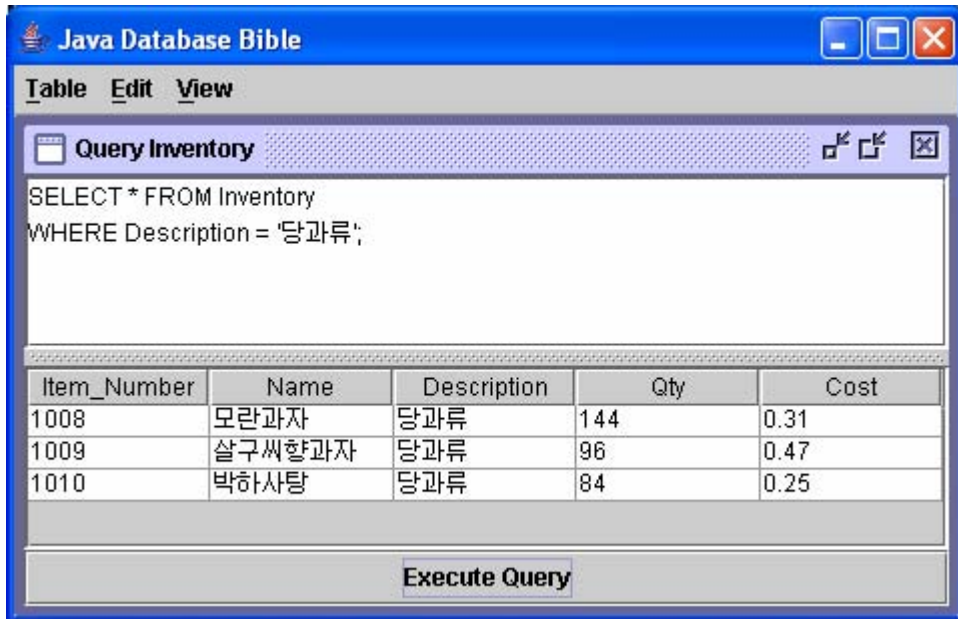



그림 4-3. SQL질문판

View차림표

[View]차림표는 DBMenu클라스를 확장한것으로서 ResultSet를 위한 DBMenuItem들을 추가하였다. 목록 4-17는 [View]차림표에 대한 코드를 보여준다.

목록 4-17. ResultSet항목을 가진 [View]차림표

```
package JavaDB_Bible.ch04.sec03;

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;

public class ViewMenu extends DBMenu{
   JMenuItem resultSetItem;
   JMenuItem scrollableResultSetItem;
   JMenuItem updatableResultSetItem;
   JMenuItem rowSetItem;

   public ViewMenu(){
      setText("View");
      setActionCommand("View");
```

```
            setMnemonic((int)'V');

            resultSetItem = new DBMenuItem("ResultSet", 'R', itemListener, false);
           scrollableResultSetItem =
                new DBMenuItem("Scrollable ResultSet", 'S', itemListener, false);
           updatableResultSetItem =
                new DBMenuItem("Updatable SesultSet", 'U', itemListener, false);
            rowSetItem = new DBMenuItem("RowSet",'W', itemListener, false);

            add(resultSetItem);
            add(scrollableResultSetItem);
            add(updatableResultSetItem);
            add(rowSetItem);
        }
    }
```

TableQueryFrame

TableQueryFrame은 1절에서 고찰했던 TableBuilderFrame과 대단히 류사하다. JInternalFrame의 확장클라스로서 ResultSet에서 돌려주는 마당들을 현시하는 JTable, 질문을 작성할수 있는 편집가능한 본문마당을 제공해주는 JTextArea, 질문을 실행시키는 <Execute Query>단추를 포함하고있다. 그밖에 이 클라스는 parseTable() 메쏘드나 TableChangeListener를 필요로 하지 않기때문에 앞에서보다 더 간단해졌다.

JTable은 아래에서 보여준것과 같이 SQL질문을 리용하여 미리 적재된다.

```
"SELECT TOP 5 * FROM " + tableName;
```

자료기지 표가 큰 경우에 비대한 JTable이 적재되는것을 막기 위해 TOP 5 제한을 리용한다. 사용자는 이것을 자기의 프로그람에 맞게 변경할수 있다.

TableQueryFrame클라스는 JTable이 아니라 JTextArea에 의해 구동되기때문에 앞에서 취급한 클라스와 대응하는 부분들이 다르다. JTextArea는 <Execute Query>단추가 눌리웠을 때 실행하는 자유형식의 SQL질문을 입력하는 곳이다.

목록 4-18에 보여준 TableQueryFrame실례의 리용에 포함된 사건들은 다음과 같다.

1. 사용자가 자료기지를 선택한다.
2. 사용자가 [View]차림표의 ResultSet를 선택한다.
3. 사용자가 표를 선택한다.
4. TableQueryFrame이 현시되면서 표에서 맨우의 5개 레코드가 나타난다.
5. SQL지령이 JTextArea에 건입력되고 지령에 따라 실행된다.

이 실례는 임의의 자료기지관리체계에 련결할수 있는 Swing에 의한 응용프로그람을 작성하기 위해 1절과 2절의 실례들을 확장한것이다. 이 실례는 표를 작성하고 그 표에 자료를 삽입하는데 리용된다. 또한 이 절에서 고찰한 임의의 질문들을 실행하는데 리용될수 있다. 목록 4-18에 TableQueryFrame코드를 보여주었다.

목록 4-18. TableQueryFrame

```
package JavaDB_Bible.ch04.sec03;

import java.awt.*;
import java.awt.event.*;
import java.util.EventObject;
import java.util.EventListener;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

class TableQueryFrame extends JInternalFrame{
  protected JTable table;
  protected JScrollPane tableScroller;
  protected JTextArea SQLPane = new JTextArea();
  protected JButton queryButton = new JButton("Execute Query");
  protected DatabaseUtilities dbUtils;

  protected String tableName = null;
  protected String colNames[] = null;
  protected String dataTypes[] = null;
  protected String SQLQuery = null;
  protected String SQLCommandRoot = "Query ";

  public TableQueryFrame(String tableName, DatabaseUtilities dbUtils){
    System.out.println(tableName+", "+dbUtils);
    setSize(500, 320);
    setLocation(10,10);
    setClosable(true);
    setMaximizable(true);
    setIconifiable(true);
    setResizable(true);
    getContentPane().setLayout(new BorderLayout());
    this.tableName = tableName;
    this.dbUtils = dbUtils;
```

```
        SQLCommandRoot = SQLCommandRoot + tableName;
        setTitle(SQLCommandRoot);
        init();
        setVisible(true);
    }

    // JInternalFrame의 초기화
    private void init(){
        colNames = dbUtils.getColumnNames(tableName);
        dataTypes = dbUtils.getDataTypes(tableName);
        SQLQuery = "SELECT TOP 5 * FROM " + tableName;
        Vector dataSet = dbUtils.executeQuery(SQLQuery);
        table = createTable(colNames, dataSet);
        JScrollPane sqlScroller = new JScrollPane(SQLPane);
        tableScroller = new JScrollPane(table);
        JSplitPane splitter = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
            sqlScroller, tableScroller);
        splitter.setDividerLocation(100);
        getContentPane().add(splitter, BorderLayout.CENTER);
        getContentPane().add(queryButton, BorderLayout.SOUTH);
        queryButton.addActionListener(new ButtonListener());
    }

    protected JTable createTable(String[] colNames, Vector dataSet){
        int nRows = dataSet.size();
        String[][] rowData = new String[nRows][colNames.length];

        for(int i=0;i<nRows;i++){
            Vector row = (Vector)dataSet.elementAt(i);
            for (int j=0;j<row.size();j++)
                rowData[i][j] = ((Object)row.elementAt(j)).toString();
        }
        JTable table = new JTable(rowData,colNames);
        return table;
    }

    // QueryButton에 대한 감시자
    class ButtonListener implements ActionListener{
        public void actionPerformed(ActionEvent event){
            SQLQuery = SQLPane.getText();
            JViewport viewport = tableScroller.getViewport();
            viewport.remove(table);
```

```
            colNames = dbUtils.getColumnNamesUsingQuery(SQLQuery);
            Vector dataSet = dbUtils.executeQuery(SQLQuery);
            table = createTable(colNames, dataSet);
            viewport.add(table);
        }
    }
}
```

DBManager클라스에 대한 변경(목록 4-19)은 차림표에 사건가로채기를 추가한것밖에 없다.

목록 4-19. DBManager

```
package JavaDB_Bible.ch04.sec03;

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;

public class DBManager extends JFrame{
    JMenuBar menuBar = new JMenuBar();
    JDesktopPane desktop = new JDesktopPane();
    String database = null;
    String tableName = null;
    String menuSelection = null;

    TableBuilderFrame tableMaker = null;
    TableEditFrame tableEditor = null;
    TableQueryFrame tableQuery = null;   // 추가된 부분
    DatabaseUtilities dbUtils = null;

    TableMenu tableMenu = new TableMenu();
    EditMenu editMenu = new EditMenu();
    ViewMenu viewMenu = new ViewMenu();   // 추가된 부분

    MenuListener menuListener = new MenuListener();

    public DBManager(){
        setJMenuBar(menuBar);
        setTitle("Java Database Bible");
```

```
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(desktop,BorderLayout.CENTER);
    setSize(new Dimension(550, 400));

    menuBar.add(tableMenu);
    tableMenu.setMenuListener(menuListener);

    menuBar.add(editMenu);
    editMenu.setMenuListener(menuListener);

    menuBar.add(viewMenu);   // 추가된 부분
    viewMenu.setMenuListener(menuListener);

    setFont(new Font("Dialog", Font.PLAIN, 18));
    setVisible(true);
    Font font = getGraphics().getFont();
    System.out.println(font);
  }

  private void displayTableBuilderFrame(){
    tableName = JOptionPane.showInputDialog(this, "Table: ",
      "Select table", JOptionPane.QUESTION_MESSAGE);
    tableMaker = new TableBuilderFrame(tableName);
    tableMaker.setCommandListener(new CommandListener());
    desktop.add(tableMaker);
    tableMaker.setVisible(true);
  }

  private void displayTableEditFrame(){
    tableName = JOptionPane.showInputDialog(this,"Table:",
       "Select table",JOptionPane.QUESTION_MESSAGE);
    tableEditor = new TableEditFrame(tableName, dbUtils);
    desktop.add(tableEditor);
    tableEditor.setVisible(true);
 }

  private void displayTableQueryFrame(){  // 추가된 부분
    tableName = JOptionPane.showInputDialog(this,"Table:",
      "Select table",JOptionPane.QUESTION_MESSAGE);
    tableQuery = new TableQueryFrame(tableName,dbUtils);
    desktop.add(tableQuery);
    tableQuery.setVisible(true);
```

```
}

private String[] parseKeyValueString(String kvString){
   String[] kvPair = null;

   int equals = kvString.indexOf("=");
   if(equals>0){
      kvPair = new String[2];
      kvPair[0] = kvString.substring(0,equals).trim();
      kvPair[1] = kvString.substring(equals+1).trim();
   }
   return kvPair;
}

private void selectDatabase(){
   database = JOptionPane.showInputDialog(this,"Database: ",
      "Select database", JOptionPane.QUESTION_MESSAGE);
   dbUtils = new DatabaseUtilities();
   dbUtils.setDatabaseName(database);
   dbUtils.setExceptionListener(new ExceptionListener());

   tableMenu.enableMenuItem("New Table",true);
   tableMenu.enableMenuItem("Drop Table",true);

   editMenu.enableMenuItem("Insert" ,true);
   editMenu.enableMenuItem("Update", true);
   editMenu.enableMenuItem("Delete", true);

   viewMenu.enableMenuItem("ResultSet",true);
}

private void executeSQLCommand(String SQLCommand){
   dbUtils.execute(SQLCommand);
}

private void dropTable(){
   tableName = JOptionPane.showInputDialog(this, "Table: ",
      "Select table",JOptionPane.QUESTION_MESSAGE);
   int option = JOptionPane.showConfirmDialog(null,
      "Dropping table " +tableName,
      "Database "+database,
      JOptionPane.OK_CANCEL_OPTION);
   if(option==0){
```

```
         executeSQLCommand("DROP TABLE "+tableName);
      }
   }

   class MenuListener implements ActionListener{
      public void actionPerformed(ActionEvent event) {
         String menuSelection = event.getActionCommand();
         if(menuSelection.equals("Database")){
            selectDatabase();
         }else if(menuSelection.equals("New Table")){
            displayTableBuilderFrame();
         }else if(menuSelection.equals("Drop Table")){
            dropTable();
         }else if(menuSelection.equals("Insert")){
            displayTableEditFrame();
         }else if(menuSelection.equals("ResultSet")){ // 추가된 부분
            displayTableQueryFrame();
         }else if(menuSelection.equals("Exit")){
            System.exit(0);
         }
      }
   }

   class ExceptionListener implements ActionListener{
      public void actionPerformed(ActionEvent event){
         String exception = event.getActionCommand();
         JOptionPane.showMessageDialog(null, exception,
            "SQL Error", JOptionPane.ERROR_MESSAGE);
      }
   }

   class CommandListener implements ActionListener{
      public void actionPerformed(ActionEvent event){
         String SQLCommand = event.getActionCommand();
         executeSQLCommand(SQLCommand);
      }
   }

   public static void main(String args[] ) {
      DBManager dbm = new DBManager();
   }
}
```

이제는 질문을 실행하기 위한 필수적인 JDBC코드를 추가하는것만이 남았다. 이 부분은 다음 소절에서 취급한다.

4.3.3. JDBC 코드

목록 4-20의 DatabaseUtilities클라스의 확장판본에서 executeQuery (String SQLQuery)메쏘드는 표의 행자료들을 포함하고있는 벡토르들중 하나의 벡토르를 돌려주기 위해 추가되였다. 벡토르들중에서 하나의 벡토르를 선택하는것은 한편으로 executeQuery()메쏘드가 제공하는 고유한 유연성을 보여주기 위한것이고 또 다른 한편으로는 목록 4-16과 약간 차이나는 방법을 증명하기 위해서이다.

getColumnNamesUsingQuery(String SQLCommand)메쏘드도 역시 추가되였는데 이 메쏘드는 전체 표의 모든 렬이름이 아니라 질문과 관련한 렬이름들에 대한 String배렬을 돌려준다.

목록 4-20. DatabaseUtilities

```
package JavaDB_Bible.ch04.sec03;

import java.awt.event.*;
import java.sql.*;
import java.util.Vector;
import sun.jdbc.odbc.JdbcOdbcDriver;

public class DatabaseUtilities{
   static String jdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver";
   static String dbName = "Contacts";
   static String urlRoot = "jdbc:odbc:";
   private ActionListener exceptionListener = null;

   public DatabaseUtilities(){
      registerDriver();
   }

   public void setDatabaseName(String dbName){
      this.dbName=dbName;
   }

   public void registerDriver(){
      try {
         Class.forName(jdbcDriver);
         DriverManager.registerDriver(new JdbcOdbcDriver());
      }
```

```
    catch(ClassNotFoundException e){
      reportException(e.getMessage());
    }
    catch(SQLException e){
      reportException(e.getMessage());
    }
  }

  public void execute(String SQLCommand){
    String url = urlRoot + dbName;
    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      stmt.execute(SQLCommand);
      con.close();
    }
    catch (SQLException e){
      reportException(e.getMessage());
    }
  }

  public void execute(String[] SQLCommand){
    String url = urlRoot + dbName;
    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      for (int i = 0;i<SQLCommand.length;i++) {
        stmt.execute(SQLCommand[i]);
      }
      con.close();
    }
    catch(SQLException e){
      reportException(e.getMessage());
    }
  }

  public String[] getColumnNames(String tableName){
    Vector dataSet = new Vector();
    String[] columnNames = null;
    String url = urlRoot + dbName;
    String SQLCommand = "SELECT * FROM "+tableName+";";

    try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(SQLCommand);
```

```
        ResultSetMetaData md = rs.getMetaData();

        columnNames = new String[md.getColumnCount()];
        for(int i=0;i<columnNames.length;i++){
            columnNames[i] = md.getColumnLabel(i+1);
        }
        con.close();
    }
    catch(SQLException e){
        reportException(e.getMessage());
    }
    return columnNames;
}

public String[] getColumnNamesUsingQuery(String SQLCommand){
    Vector dataSet = new Vector();
    String[] columnNames = null;
    String url = urlRoot+dbName;

    try {
        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(SQLCommand);
        ResultSetMetaData md =  rs.getMetaData();

        columnNames = new String[md.getColumnCount()];
        for (int i=0;i<columnNames.length;i++){
            columnNames[i] = md.getColumnLabel(i+1);
        }
        con.close();
    }
    catch(SQLException e){
        reportException(e.getMessage());
    }
    return columnNames;
}

public String[] getDataTypes(String tableName){
    Vector dataSet = new Vector();
    String[] dataTypes = null;
    String url = urlRoot + dbName;
    String SQLCommand = "SELECT * FROM "+tableName+";";

    try{
        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();
```

```
      ResultSet rs = stmt.executeQuery(SQLCommand);
      ResultSetMetaData md = rs.getMetaData();

      dataTypes = new String[md.getColumnCount()];
      for (int i=0;i<dataTypes.length;i++) {
         dataTypes[i] = md.getColumnTypeName(i+1);
      }
      con.close();
   }
   catch(SQLException e){
      reportException(e.getMessage());
   }
   return dataTypes;
}

public Vector executeQuery(String SQLQuery){
   Vector dataSet = new Vector();
   String url = urlRoot + dbName;

   try {
      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(SQLQuery);
      ResultSetMetaData md = rs.getMetaData();

      int nColumns = md.getColumnCount();
      while (rs.next()) {
         Vector rowData = new Vector();
         for (int i=1;i<=nColumns;i++){
            rowData.addElement(rs.getObject(i));
         }
         dataSet.addElement(rowData);
      }
      con.close();
   }
   catch(SQLException e){
      reportException(e.getMessage());
   }
   return dataSet;
}

public void setExceptionListener(ActionListener exceptionListenner){
   this.exceptionListener = exceptionListener;
}

private void reportException(String exception){
```

```
      if(exceptionListener!=null){
        ActionEvent evt = new ActionEvent(this,0,exception);
        exceptionListener.actionPerformed(evt);
      }else{
        System.err.println(exception);
      }
    }
  }
```

ORDER BY, GROUP BY와 같은 SQL질문들을 리용하여 탐색결과를 정리할수 있다.

제4절. 탐색결과의 조직과 색인의 리용

이 절에서는 SQL질문에 의해 귀환된 자료를 정리하고 분석하는 여러가지 방법들을 론의한다. 이 방법들에는 자료의 정렬과 그룹화, 통계분석의 실행, 그리고 그룹화된 결과들의 려과가 포함된다. 이 절에서는 또한 색인을 리용한 질문의 효과성에 대해서도 구체적인 실례를 들어 설명한다.

4.4.1. 결과자료의 정렬, 그룹화, 통계분석, 려과

ORDER BY문절을 리용한 질문결과의 정렬

다음 실례는 Customers표에서 《성철》이라는 이름을 가진 모든 주문자들을 검색하고 성의 자모순에 따라 올리순서로 정렬하는 코드이다. 기정의 정렬순서는 올리순서이지만 DESC열쇠단어를 추가하여 내리순서로 바꿀수 있다.

```
SELECT Family_Name, Personal_Name, City, State
FROM Costomers
WHERE Personal_Name = '성철'
ORDER BY Family_Name;
```

여러개 렬을 정렬하려면 정렬목록을 리용한다. 실례로 Personal_Name렬에 기초해서 올리순서로 자료를 정렬한 다음 Family_Name렬에 의하여 내리순서로 다시 정렬하려면 다음과 같은 정렬목록을 리용한다.

```
ORDER BY Personal_Name, Family_Name DESC;
```

ORDER BY문절을 리용하지 않을 때 질문의 출력순서는 정해지지 않는다.

주의: DatabaseMetaData객체는 NULL들에 대한 정렬순서를 결정하기 위한 일련의 메쏘드들을 제공한다:

```
boolean nullsAreSortedAtStart()
boolean nullsAreSortedAtEnd()
```

GROUP BY문절

GROUP BY문절은 다음 실례에 보여준것처럼 지적한 마당에서 동일한 값들을 가진 레코드들을 하나의 레코드로 결합한다.

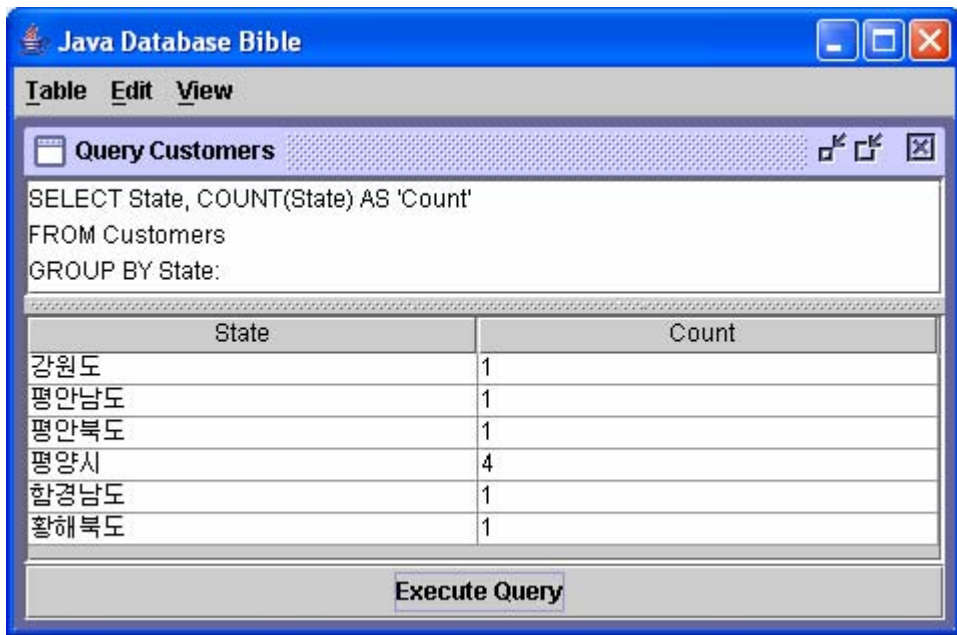



그림 4-4. 도별 주문자들의 수를 얻기 위한 GROUP BY문절

GROUP BY문절은 한개 렬에서 같은 값을 가진 모든 레코드들을 단일한 레코드로 결합하기때문에 SELECT문절에 렬거된 매개 렬이름들은 GROUP BY문절에 지적된 렬이거나 COUNT()나 SUM()와 같은 렬함수여야 한다.

ORDER BY문절에서 하나이상의 렬들을 리용할수 있은것과 같이 하나이상의 렬들을 그룹화할수 있다. 그림 4-5에서는 하나이상의 렬들을 그룹화하는 실례를 보여준다.

주의: SELECT문에서 지적된 모든 렬이름은 GROUP BY문절에서도 언급되여야 한다. 어느쪽에도 언급되지 않은 렬이름은 오유로 된다. GROUP BY문절은 GROUP BY문절속의 Description과 State의 매개 유일한 렬조합에 대하여 한개의 행을 돌려준다.

GROUP BY문절은 중요하게 자료분석을 목적으로 자료를 그룹화한다. 자료그룹들을 분석하는데 리용되는 함수들이 집계함수이다.

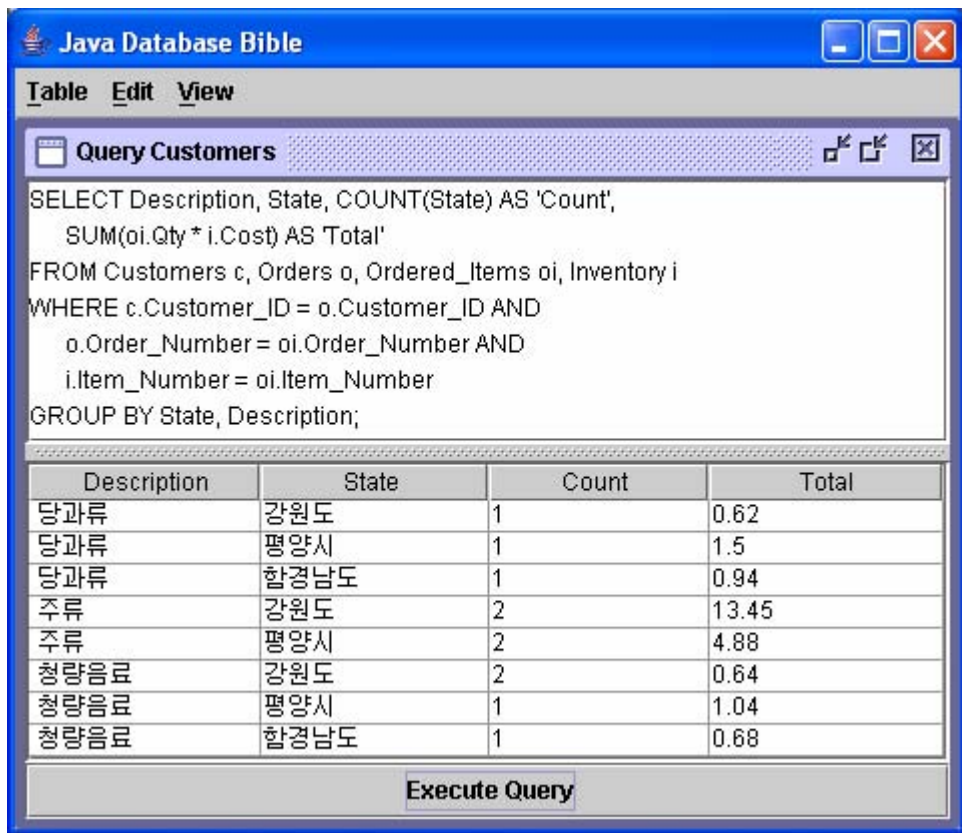



그림 4-5. 여러개 렬들에 대한 GROUP BY의 리용

집계함수

집계함수는 자료의 렬에 대한 연산으로부터 단일한 결과값을 돌려준다. 이 함수들은 개별적인 자료요소들에 대해 작용하는 산수함수, 론리함수, 문자함수와 구별된다. 대부분의 자료기지관리체계들은 SUM, AVG, COUNT, STDEV, MAX, MIN과 같은 집계함수들을 지원한다.

집계함수들은 자료요소들의 그룹에 대한 통계정보나 요약정보를 제공한다. 이 그룹들

은 GROUP BY문절에 의해 특별히 작성될수도 있지만 집계함수가 전체결과모임인 기정의 그룹에 적용될수도 있다.

집계함수들의 가장 실천적인 리용실례는 단순한 판매보고서를 작성하는것이다. 그림 4-6의 질문에서는 서로 다른 주문자들을 라렬하고 그 주문자들이 구매한 상품들의 수와 총가격을 계산하는 결과모임을 작성한다.

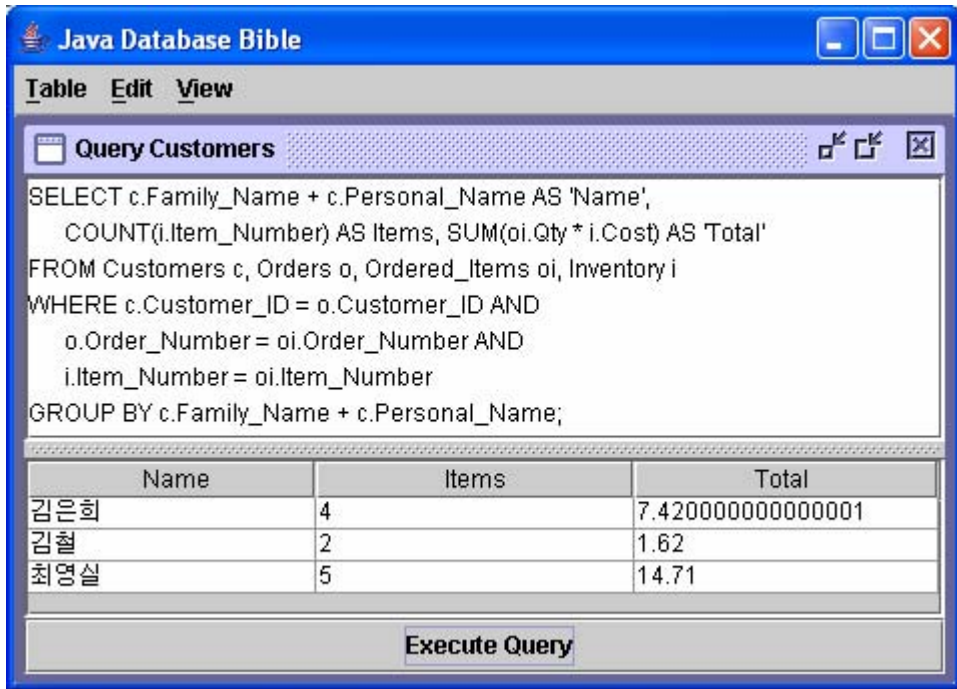



그림 4-6. 주문자들의 주문상품개수와 총금액을 얻기 위한 집계함수들의 리용

이 실례에서는 주문자에 의해 주문자료를 그룹화하였으므로 결과모임의 매 행은 주문자정보가 현시될수 있도록 한명의 주문자를 나타낸다. 집계함수들은 주문자들이 구입한 모든 품목들에 작용하므로 그것들도 역시 SELECT목록에 포함될수 있다.

HAVING문절을 리용한 그룹의 려과

HAVING문절은 적절한 조건을 그룹들에 적용하여 자료기지관리체계가 조건을 만족시키는 그룹들에 해당한 결과만을 돌려줄수 있게 해준다. 그룹들을 려과하려면 GROUP BY문절다음에 HAVING문절을 적용한다. 그림 4-7은 HAVING문절을 리용하여 도별로 주문자수

를 계산하고 주문자가 한명뿐인 도들을 려과하는 실례를 보여주고있다.

Description	State	Count	Total
주류	강원도	2	13.45
주류	평양시	2	4.88
청량음료	강원도	2	0.64

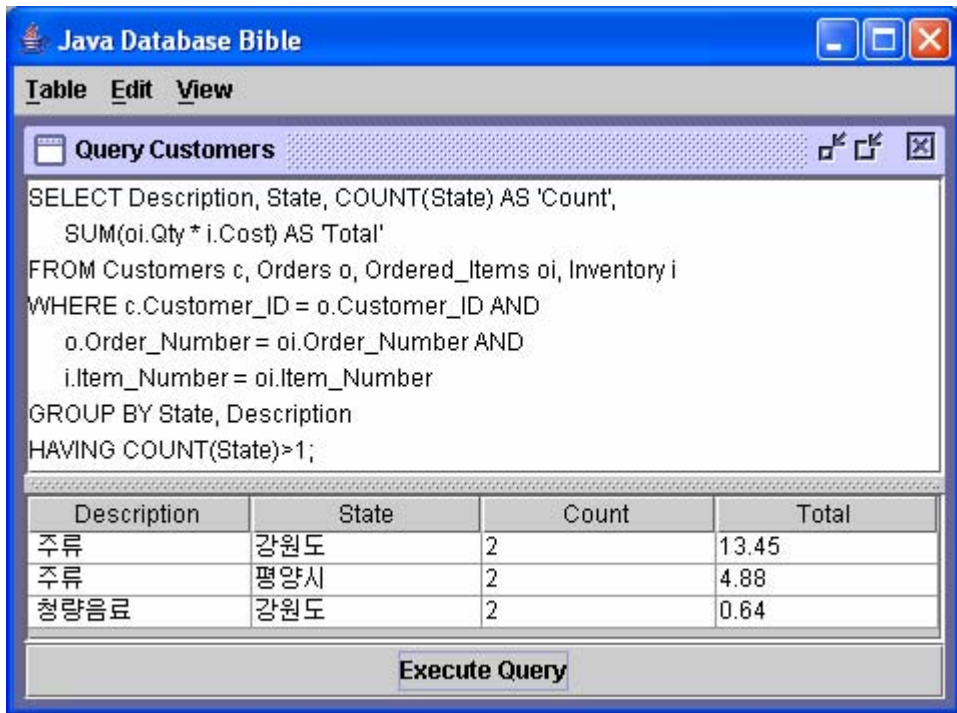



그림 4-7. HAVING문절의 리용

GROUP BY를 생략하여 전체 결과모임에 HAVING문절을 적용할수도 있다. 이 경우에 자료기지관리체계는 전체표를 한개의 그룹으로 취급하므로 기껏해서 한개의 결과행이 있게 된다. 만약 HAVING조건이 전체 표에 대해 참이 아니면 한개의 행도 반환되지 않는다.

HAVING문절은 AND와 OR에 의해 련결된 한개이상의 술어부들을 포함할수 있다. 매개 술어부는 그룹의 속성(COUNT(State)와 같은)을 그룹의 다른 속성이나 어떤 상수와 비교한다.

HAVING문절과 WHERE문절은 둘다 질문에서 다양한 려과를 실시할수 있게 해준다. 주되는 차이는 HAVING문절이 귀환된 결과모임안에 있는 그룹들에 적용된다는데 대하여 WHERE문절은 SELECT문의 대상을 이루고있는 전체표 또는 표들의 그룹에 적용된다는것이다.

4.4.2. 색인을 리용한 SQL질문의 효과성제고

색인들을 리용하면 자료기지의 성능을 현저히 개선할수 있다. **색인**(index)은 표나 뷰에서 어떤 항목들을 찾아보는 빠른 방법을 제공하는 구조이다. 사실상 색인은 표나 뷰에 있는 행들에 대한 순서화된 지적자배렬이라고 할수 있다.

매 행에 대한 유일한 id를 열쇠로 배당하면 그 표에 대한 색인을 미리 정의한것과 같다. 이것은 id렬에 기초하여 표들이 결합(join)되여있을 때 일반적으로 요구되는 id에 의한 항목찾기를 자료기지관리체계가 매우 빨리 할수 있게 해준다.

CREATE INDEX문은 요구되는 렬이나 렬들의 그룹에 해당한 색인을 추가할수 있게 해준다. 례를 들어 사용자가 도별로 탐색을 하려고 할 때 유일한 행 id는 아무런 도움도 주지 못하며 자료기지관리체계는 질문에 맞는 모든 도이름을 찾기 위해 전체 표에 대한 맹목적인 탐색을 진행해야 한다. 만약 도이름에 의한 질문을 많이 하려고 계획하였다면 도이름렬에 색인을 추가해야 한다. 만약 그렇지 않으면 사용자는 자모순으로 배렬되지 않은 전화번호수첩을 찾아보는 경우와 같은 처지에 놓일것이다.

색인을 추가하기 위한 SQL지령의 사용법은 다음과 같다.

```
CREATE INDEX STATE_INDEX ON Contact_Info(STATES);
```

CREATE INDEX열쇠단어 다음에 색인에 대한 이름을 규정하고 표이름과 색인으로 되는 렬목록을 정의한다. 색인을 제거하려면 DROP INDEX지령을 사용한다.

```
DROP INDEX Contact_Info.STATE_INDEX;
```

주의: DROP INDEX지령에서 색인이름을 지적하는것은 SQL의 변종에 따라 다르므로 리용할 때에는 자기가 리용하는 자료기지관리체계의 지도서를 참고해야 한다. 그렇지 않으면 SQL례외가 발생한다.

색인이름은 그 앞에 색인이 적용되는 표이름을 붙여 완전히 정의되여야 한다는데 대하여 주의하기 바란다.

목록 4-21의 실례는 질문의 경과시간을 계산하여 색인리용의 효과성을 보여주는 추가적인 코드들을 포함하고있는 간단한 JDBC이다.

목록 4-21. 색인의 작성과 삭제

```
package JavaDB_Bible.ch04.sec04;

import java.sql.*;

public class PrintIndexedResultSet{
   public static void main(String args[]){
      String query =
```

```
        "SELECT STATE, COUNT(STATE) FROM " +
        "Contact_Info GROUP BY STATE";
      PrintIndexedResultSet p = new PrintIndexedResultSet(query);
   }

   public PrintIndexedResultSet(String query){
      try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(
           "jdbc:odbc:Contacts");
        Statement stmt = con.createStatement();
        stmt.executeUpdate("CREATE INDEX STATE_INDEX ON "+
           "Contact_Info(STATE)");

        java.util.Date startTime = new java.util.Date();

        ResultSet rs = stmt.executeQuery(query);
        ResultSetMetaData md  = rs.getMetaData();

        int nColumns = md.getColumnCount();
        for(int i=1;i<=nColumns;i++) {

System.out.print(md.getColumnLabel(i)+((i==nColumns)?"\n":"\t"));
        }

        while(rs.next()) {
           for(int i=1;i<=nColumns;i++) {

System.out.print(rs.getString(i)+((i==nColumns)?"\n":"\t"));
           }
        }
        java.util.Date endTime = new java.util.Date();
        long elapsedTime = endTime.getTime() - startTime.getTime();
        System.out.println("Elapsed time: "+elapsedTime);

        stmt.executeUpdate("DROP INDEX Contact_Info.STATE_INDEX");
      }
      catch(ClassNotFoundException e){
        e.printStackTrace();
      }
      catch(SQLException e){
        System.out.println("Elapsed time: ");
        e.printStackTrace();
      }
   }
}
```

CREATE INDEX와 DROP INDEX행들을 제거하고 실례를 실행하면 색인을 추가하는것이 속도개선에 얼마만한 영향을 주는가를 쉽게 알수 있다. 목록 4-21의 실례가 약 150,000명의 회원들을 포함하고있는 회원자료기지에 대하여 실행될 때에는 질문을 수행하는데 드는 경과시간은 색인을 달지 않았을 때의 절반으로 줄어든다.

제5절. 2층모형의 구축

이 절에서는 JDBC 핵심 API에 대한 론의를 심화시키면서 의뢰기/봉사기방식의 간단한 구성요소들을 사용하는 코드실례들을 배비한다. 그외에 범용적인 자료기지관리조종탁프로그람을 작성하기 위하여 이 실례들을 결합시킨다. 이 응용프로그람은 평문파일로부터 완전한 객체관계형자료기지까지 임의의 자료원천과 작업하기 위한 일반적인 도구일식의 기초를 형성한다. 취급하는 내용들은 다음과 같다.

- 각이한 자료기지와 구동프로그람의 리용
- DatabaseMetaData의 리용
- ResultSetMetaData의 리용

4.5.1. 각이한 자료기지와 구동프로그람의 사용

JDBC의 유연성을 론증하기 위하여 다양한 관계형자료기지관리체계에서 Contacts자료기지의 복사판들을 만들고 그것들을 JComboBox에 하나의 목록으로 보여줄수 있다. JComboBox는 그림 4-8에서 보는것처럼 자료기지선택에 리용되는 JOptionPane에 현시된다.

이 새로운 대화칸을 리용하면 사용자는 다양한 Contacts시험자료기지들가운데서 임의의것을 선택할수 있다. 일단 자료기지이름이 선택되면 JDBC구동프로그람을 선택할수 있게 하는 두번째 대화칸이 현시된다.

다른 구동프로그람들의 사용에 대하여 1절에서 간단히 언급하였지만 앞에서의 실례들은 모두 자료기지관리체계의 선택을 미결로 남겨두고 JDBC-ODBC다리를 사용하였다. 리유는 실지 자료기지를 만들고 작업하는데로 직접 들어가기 위해서이다.

3장에서 언급한것처럼 JDBC-ODBC다리는 다양한 자료기지와 작업할 때 하나의 중요한 우점을 가진다. 즉 JDBC-ODBC다리는 거의 임의의 관계형자료기지관리체계와 함께 리용될수 있다. 그러나 대부분의 다른 구동프로그람들은 특정한 자료기지체계에만 해당된다. 한편 JDBC-ODBC다리의 결합은 특정한 관계형자료기지관리체계에 대하여 최적화된 순수한 Java구동프로그람보다 효률성이 낮다는것이다.

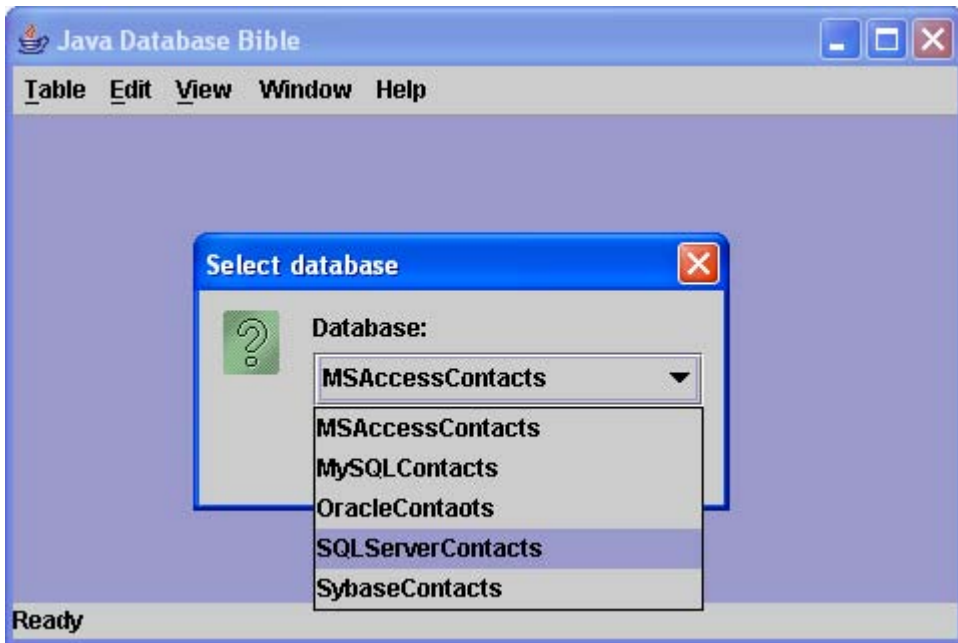



그림 4-8. JComboBox를 리용하여 각이한 자료기지를 선택

DriverManager는 JDBC구동프로그람을 두가지 방법으로 적재할수 있다.

- 초기화시 DriverManager가 jdbc.drivers체계속성에 렬거되여있는 구동프로그람들을 적재할수 있다.
- 임의의 시간에 프로그람이 Class.forName()을 사용하여 JDBC구동프로그람들을 명시적으로 적재할수 있다.

getConnection()이 호출되면 DriverManager는 초기화시에 적재된것들과 명시적으로 적재된 구동프로그람들속에서 적합한 구동프로그람을 찾는다. 그러기 위하여 DriverManager는 매 구동프로그람의 acceptsURL()메쏘드에 자료기지의 URL을 넘겨주면서 등록된 모든 구동프로그람들에 개별적으로 문의한다.

JDBC구동프로그람의 명시적인 적재를 설명하기 위하여 각이한 JDBC들이 렬거되여있는 JComboBox를 가진 새로운 JOptionPane을 DBManager클라스에 추가하였다. 따라서 사용자는 자료기지관리체계 SQL Server를 선택한 다음 JdbcOdbcDriver나 Opta2000 pure자바구동프로그람중에서 어느것이나 사용할수 있게 한다.

프로그람에는 또한 다음의 기능들을 추가하기 위한 몇가지 새로운 특징들이 포함되여있다.

- 사용자가 결과모임들과 기타 정보를 현시하는데 리용되는 JInternalFrames를 계단식이나 타일식으로 정렬시키는것과 같은 창문관리과제들을 수행하게 하는 일부 지원코드를 가진 창문차림표(WindowMenu)가 추가되였다.
- 사용자가 사용중에 있는 자료기지관리체계와 JDBC구동프로그람에 대한 정보에 접근할수 있게 하는 도움말차림표(HelpMenu)가 추가되였다.
- 통보문과 문실행에 요구되는 시간을 보여주기 위해 JFrame의 밑부분에 상태판(StatusPanel)이 추가되였다.
- 자료기지에 접속하기전 체계시간과 접속한 다음의 체계시간을 얻기 위한 코드가 추가되였다. 단위가 ms인 경과시간은 체계접속시작시간과 마지막시간의 차로 계산되여 JFrame의 밑에 추가된 상태판에 현시된다.

창문차림표와 도움말차림표의 코드는 이 장의 앞부분에서 본 차림표들과 류사하다. 목록 4-22는 지원된 계단과 타일기능을 보여준다.

목록 4-22. WindowMenu클라스

```
package JavaDB_Bible.ch04.sec05;

import java.awt.*;
import javax.swing.*;

public class WindowMenu extends DBMenu{
   public WindowMenu(){
      setText("Window");
      setActionCommand("Window");
      setMnemonic((int)'W');
      setBorderPainted(false);

      add(new DBMenuItem("Cascade", 'C', itemListener,true));
      add(new DBMenuItem("Tile horizontally", 'H', itemListener, true));
      add(new DBMenuItem("Tile vertically",'V', itemListener, true));
   }
}
```

창문관리기능들은 DBManager클라스에서 cascade(), tileVertically(), tileHorizontally()메쏘드들을 통하여 수행된다. 한편 selected()메쏘드는 JInternalFrame이 정확히 자리잡도록 하기 위하여 현재 선택된 JInternalFrame을 식별하는데 리용된다.

StatusPanel클라스도 역시 매우 간단하다. 목록 4-23에서 보여준것처럼 BorderLayout를 가진 JPanel의 CENTER와 EAST령역에 추가된 몇개의 JLabel들을 병합한다. StatusPanel은 기본 JFrame의 SOUTH부분에 추가된다. JavaBean의 스타일-설정자메쏘드들이 StatusPanel이 현시하는 통보문들을 설정하는데 리용된다.

목록 4-23. StatusPanel

```
package JavaDB_Bible.ch04.sec05;

import java.awt.*;
import javax.swing.*;

public class StatusPanel extends JPanel{
   JLabel msgLabel = new JLabel();
   JLabel timerLabel = new JLabel();
   public StatusPanel(){
      setLayout(new BorderLayout());
      add(msgLabel, BorderLayout.CENTER);
      add(timerLabel, BorderLayout.EAST);
   }

   public StatusPanel(String message){
      this();
      setMessage(message);
   }

   public void setMessage(String message){
      msgLabel.setText(message);
   }

   public void setTimerMsg(String message){
      timerLabel.setText(message);
   }
}
```

확장된 DBManager클라스

DBManager클라스는 아주 넓은 범위에서 변화되였으므로 전체 클라스를 목록 4-24에서 보여준다. 변경된 내용들은 쉽게 알아볼수 있도록 해설문을 추가하였다.

목록 4-24. DBManager클라스

```
package JavaDB_Bible.ch04.sec05;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DBManager extends JFrame {
    JMenuBar menuBar = new JMenuBar();
    JDesktopPane desktop = new JDesktopPane();
    StatusPanel statusBar = new StatusPanel("Ready");
    String database = null;
    String jdbcDriver = null;
    String tableName = null;
    String menuSelection = null;

    TableBuilderFrame tableMaker = null;
    TableEditFrame tableEditor = null;
    TableQueryFrame tableQuery = null;
    DatabaseUtilities dbUtils = null;
    InfoDialog infoDlg = null;

    TableMenu tableMenu = new TableMenu();
    EditMenu editMenu = new EditMenu();
    ViewMenu viewMenu = new ViewMenu();
    WindowMenu windowMenu = new WindowMenu();
    HelpMenu helpMenu = new HelpMenu();

    MenuListener menuListener = new MenuListener();

    public DBManager() {
        setJMenuBar(menuBar);
        setTitle("Java Database Bible");

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(desktop, BorderLayout.CENTER);
        getContentPane().add(statusBar, BorderLayout.SOUTH);
        setSize(new Dimension(480, 320));

        menuBar.add(tableMenu);
        tableMenu.setMenuListener(menuListener);
```

```
        menuBar.add(editMenu);
        editMenu.setMenuListener(menuListener);

        menuBar.add(viewMenu);
        viewMenu.setMenuListener(menuListener);

        menuBar.add(windowMenu); // 추가된 부분
        windowMenu.setMenuListener(menuListener);

        menuBar.add(helpMenu); // 추가된 부분
        helpMenu.setMenuListener(menuListener);

        setVisible(true);
    }

    private void displayTableBuilderFrame() {
        tableName = JOptionPane.showInputDialog(this, "Table:",
                                              "Select table",

JOptionPane.QUESTION_MESSAGE);
        tableMaker = new TableBuilderFrame(tableName);
        tableMaker.setCommandListener(new CommandListener());
        tableMaker.setSize(desktop.getSize());
        desktop.add(tableMaker);
        tableMaker.setVisible(true);
    }

    private void displayTableEditFrame() {
        tableName = JOptionPane.showInputDialog(this, "Table:",
                                                     "Select table",
                                          JOptionPane.QUESTION_MESSAGE);
        tableEditor = new TableEditFrame(tableName, dbUtils);
        desktop.add(tableEditor);
        tableEditor.setSize(desktop.getSize());
        tableEditor.setVisible(true);
    }

    private void displayTableQueryFrame() {
        tableName = JOptionPane.showInputDialog(this, "Table:",
                                                     "Select table",
                                          JOptionPane.QUESTION_MESSAGE);
        tableQuery = new TableQueryFrame(tableName, dbUtils);
```

```
        desktop.add(tableQuery);
        tableQuery.setSize(desktop.getSize());
        tableQuery.setVisible(true);
    }

    // 추가된 부분
    private void displayInfoDialog() {
        infoDlg = new InfoDialog(dbUtils);
        Rectangle r = getBounds();
        infoDlg.setBounds(r.x + r.width - 250, r.y + 50, 240, 240);
        infoDlg.setVisible(true);
    }

    private void selectDatabase() {
        // 수정된 부분
        String[] databases = {"MSAccessContacts", "MySQLContacts",
                              "OracleContaots", "SQLServerContacts",
                              "SybaseContacts"};
        database = (String) JOptionPane.showInputDialog(null, "Database:",
              "Select database", JOptionPane.QUESTION_MESSAGE,
              null, databases, databases[0]);
        dbUtils = new DatabaseUtilities();
        dbUtils.setDatabaseName(database);

        // 추가된 부분
        dbUtils.setJdbcDriverName(selectJDBCDriver());
        if (jdbcDriver.equals("com.inet.tds.TdsDriver"))
           dbUtils.setDatabaseUrl("jdbc:inetdae7:localhost");
        else
           dbUtils.setDatabaseUrl("jdbc:odbc:" + database);

        if (!dbUtils.connectToDatabase(database)) {
           statusBar.setMessage("Error connecting to " + database);
           return;
        }

        // 추가된 부분
        statusBar.setMessage("Retrieving MetaData from " + database);
        statusBar.repaint();

        System.out.println("Retrieving MetaData from " + database);
        java.util.Date startTime = new java.util.Date();
```

```
        MetaDataFrame dbTree = new MetaDataFrame(database, dbUtils);
        java.util.Date endTime = new java.util.Date();
        long elapsed = endTime.getTime() - startTime.getTime();
        statusBar.setTimerMsg("Elapsed time = " + elapsed + " ms");

        desktop.add(dbTree);
        dbTree.setSize(desktop.getSize());
        dbTree.setVisible(true);

        tableMenu.enableMenuItem("New Table", true);
        tableMenu.enableMenuItem("Drop Table", true);

        editMenu.enableMenuItem("Insert", true);
        editMenu.enableMenuItem("Update", true);
        editMenu.enableMenuItem("Delete", true);

        viewMenu.enableMenuItem("ResultSet", true);
        helpMenu.enableMenuItem("Database Info", true);
    }

    // 추가된 부분
    private String selectJDBCDriver() {
        String[] drivers = {"sun.jdbc.odbc.JdbcOdbcDriver",
                            "com.inet.tds.TdsDriver"};
        jdbcDriver = (String) JOptionPane.showInputDialog(null,
                     "JDBCDriver;", "Select JDBC Driver",
                     JOptionPane.QUESTION_MESSAGE, null,
                     drivers, drivers[0]);
        return jdbcDriver;
    }

    private void executeSQLCommand(String SQLCommand) {
        dbUtils.execute(SQLCommand);
    }

    private void dropTable() {
        tableName = JOptionPane.showInputDialog(this, "Table:",
                                                "Select table",

JOptionPane.QUESTION_MESSAGE);
        if (tableName != null && tableName != "") {
```

```
        int option = JOptionPane.showConfirmDialog(null,
              "Dropping table " + tableName, "Database " +
              database, JOptionPane.OK_CANCEL_OPTION);
        if (option == 0) {
           executeSQLCommand("DROP TABLE " + tableName);
        }
     }
  }

  // 추가된 부분
  private int selected() {
     JInternalFrame[] jif = desktop.getAllFrames();
     for (int i = 0; i < jif.length; i++) {
        if (jif[i].isSelected())return i;
     }
     return 0;
  }

  // 추가된 부분
  private void cascade() {
     JInternalFrame[] jif = desktop.getAllFrames();
     int j = selected();
     int nJifs = jif.length;
     j = (j < nJifs - 1) ? j + 1 : 0;
     Dimension d = desktop.getSize();

     for (int i = 0; i < nJifs; i++) {
        jif[i].setBounds(new Rectangle(i * 20, i * 20,
                               d.width - nJifs * 20,
                               d.height - nJifs * 20));
        jif[i].toFront();
        j = (j < nJifs - 1) ? j + 1 : 0;
     }
  }

  // 추가된 부분
  private void tileVertically() {
     JInternalFrame[] jif = desktop.getAllFrames();
     int j = selected();
     int nJifs = jif.length;
     Dimension d = desktop.getSize();
     for (int i = 0; i < nJifs; i++) {
```

```
            jif[i].setBounds(new Rectangle(i * d.width / nJifs,
                                    0, d.width / nJifs, d.height));
            jif[i].toFront();
            j = (j < nJifs - 1) ? j + 1 : 0;
        }
    }

    // 추가된 부분
    private void tileHorizontally() {
        JInternalFrame[] jif = desktop.getAllFrames();
        int j = selected();
        int nJifs = jif.length;
        Dimension d = desktop.getSize();
        for (int i = 0; i < nJifs; i++) {
            jif[i].setBounds(new Rectangle(0,
                                    i * d.height / nJifs, d.width,
                                    d.height / nJifs));
            jif[i].toFront();
            j = (j < nJifs - 1) ? j + 1 : 0;
        }
    }

    class MenuListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            String menuSelection = event.getActionCommand();
            if (menuSelection.equals("Database")) {
                selectDatabase();
            } else if (menuSelection.equals("New Table")) {
                displayTableBuilderFrame();
            } else if (menuSelection.equals("Drop Table")) {
                dropTable();
            } else if (menuSelection.equals("Insert")) {
                displayTableEditFrame();
            } else if (menuSelection.equals("ResultSet")) {
                displayTableQueryFrame();
            } else if (menuSelection.equals("Cascade")) {
                // 추가된 부분
                cascade();
            } else if (menuSelection.equals("Tile vertically")) {
                tileVertically();
            } else if (menuSelection.equals("Tile horizontally")) {
                tileHorizontally();
```

```
            } else if (menuSelection.equals("Database Info")) {
               displayInfoDialog();
            } else if (menuSelection.equals("Exit")) {
               System.exit(0);
            }
         }
      }

      class ExceptionListener implements ActionListener {
         public void actionPerformed(ActionEvent event) {
            String exception = event.getActionCommand();
            JOptionPane.showMessageDialog(null, exception,
                                          "SQL Esror",
                                          JOptionPane.ERROR_MESSAGE);
         }
      }

      class CommandListener implements ActionListener {
         public void actionPerformed(ActionEvent event) {
            String SQLCommand = event.getActionCommand();
            executeSQLCommand(SQLCommand);
         }
      }

      public static void main(String args[]) {
         DBManager dbm = new DBManager();
      }
   }
```

서로 다른 자료기지와 각이한 구동프로그람들에서 JDBC응용프로그람을 리용하는 간단한 방법은 다음 소절에서 보기로 한다.

4.5.2. DatabaseMetaData의 리용

DatabaseMetaData대면부가 제공하는 자료기지에 대한 정보들은 다음과 같다.

- 자료원천에 대한 일반적인 정보
    - 자료기지제품이름과 판본
    - 구동프로그람이름
    - 자료기지URL

- 지원특징
  - SQL92 지원준위
  - 인정된 SQL열쇠단어
  - 지원된 거래격리준위
  - 묶음갱신과 같은 특징의 지원

- 자료원천의 제한
  - 표의 최대렬개수
  - 렬이름과 표이름길이의 최대수
- 원천이 포함하는 SQL객체들에 대한 정보
  - 일람표에서 표의 류형
  - 매개 류형에 따르는 모든 표들의 이름
  - 표들에 있는 모든 렬들에 대한 정보

대부분의 DatabaseMetaData메쏘드들이 ResultSet에 정보를 돌려주며 이 정보검색을 위하여 사용자는 getString(), getInt()와 같은 ResultSet메쏘드를 사용한다. 만일 주어진 메타자료의 형식을 리용할수 없다면 이 메쏘드들은 SQLException을 던진다. 아래에서 자료기지에 대한 정보를 검색하는 방법을 설명한다.

자료기지에 대한 정보검색

그림 4-9는 DatabaseMetaData객체를 사용하여 자료기지에 대하여 얻을수 있는 정보의 종류를 설명한다. JTree는 자료기지에서 표들의 류형을 현시하며 자식마디로 현시된 매개 류형의 표에 대한 이름들을 함께 현시한다. 표들은 렬이름들을 볼수 있게 펼칠수 있으며 렬자체도 그 렬에 대한 정보를 볼수 있도록 펼쳐질수 있다.

그림 4-9는 또한 응용프로그람이 JDBC-ODBC구동프로그람을 사용하여 현시해야 할 모든 메타자료를 얻는데 얼마만한 시간이 걸리는가를 보여준다. 경과시간이 거의 9s로서 2s를 약간 넘는 Opta2000구동프로그람과 대조된다.

대부분의 DatabaseMetaData메쏘드들은 "String pattern"이라는 인수들을 가지고 있다. 이 인수들에는 문자렬과 통용문자들이 섞여있을수 있다. 통용문자들은 SQL문자렬에서의 표준통용문자들과 일치한다. 만일 탐색패턴인수가 NULL로 설정되면 그 인수의 조건은 탐색에서 무시된다.

만일 구동프로그람이 메타자료메쏘드를 지원하지 않는다면 보통 SQL례외가 던져진다. ResultSet를 돌려주는 메쏘드들인 경우에 ResultSet(혹시 비여있을수도 있는)를 돌려줄수도 있고 SQL례외를 던질수도 있다.

사용자가 자료기지에 접속하면 DatabaseMetaData객체는 이 자료기지에 속해있는 표들의 류형과 그 표들에 속해있는 렬들 그리고 매 렬자체에 대한 정보까지 포함한다. 결과들은 JTree에 표시된다.

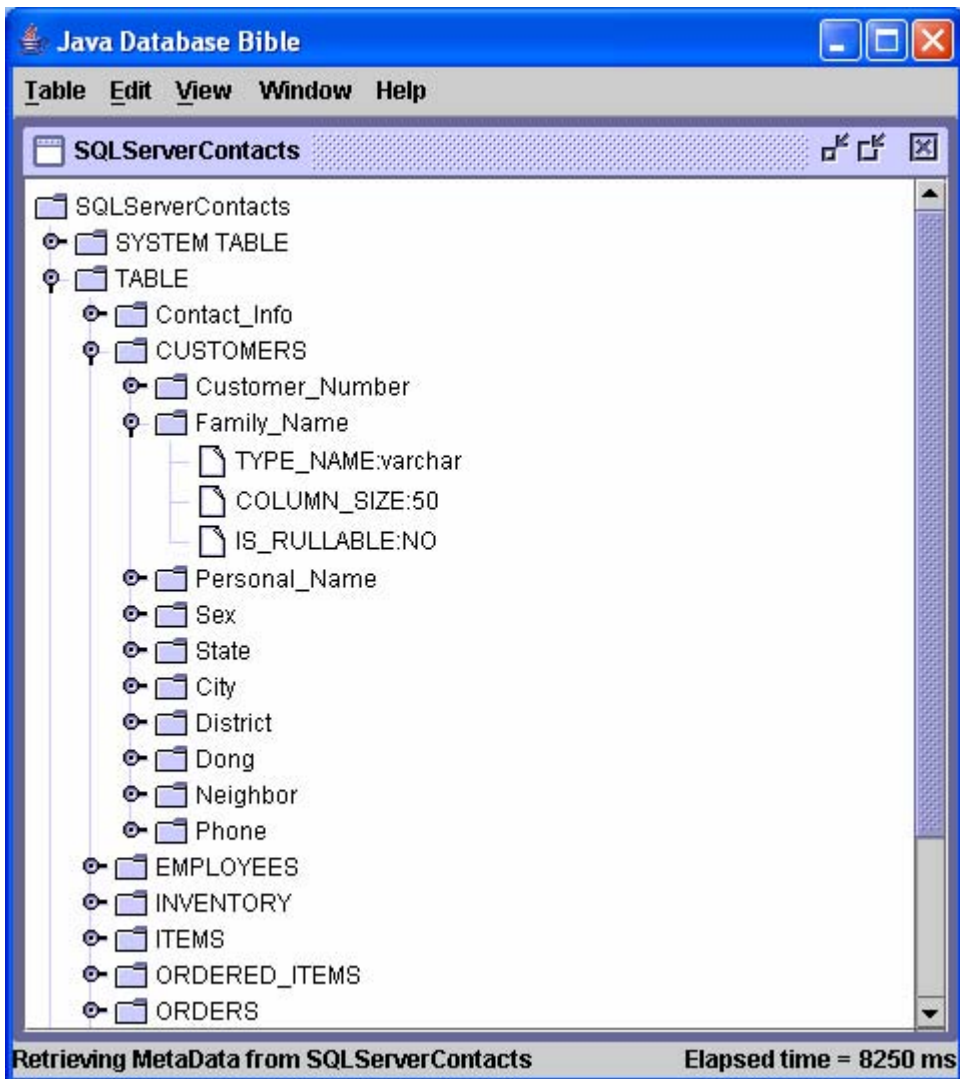



그림 4-9. 자료기지에서 표들의 나무구조보기

DatabaseMetaData객체는 Connection.getMetaData()메쏘드를 리용하여 생성된다. 자료기지에서 표의 류형들을 얻는 목록 4-25의 실례와 같이 이 객체는 자료기지에 대한 정보를 얻는데 리용된다.

목록 4-25. 표류형 검색

```
public Vector getTableTypes() {
   Vector typeVector = new Vector();
   try {
      Connection  con  =  DriverManager.getConnection(url,  userName,
password);
      DatabaseMetaData dbmd = con.getMetaData();
      ResultSet rs = dbmd.getTableTypes();
      while (rs.next()) {
         typeVector.addElement(rs.getString(1));
      }
      con.close();
   } catch (SQLException e) {
      reportException(e.getMessage());
   }
   return typeVector;
}
```

getTableTypes()메쏘드는 표의 류형을 식별하는 행마다 하나의 String렬을 포함하는 ResultSet를 돌려준다. 표의 류형에는 다음과 같은것들이 있다.

- TABLE
- VIEW
- SYSTEM TABLE

이러한 표류형정보를 리용하면 getTables()메쏘드로 실제 표이름들을 얻을수 있다. 목록 4-26에 그 실례를 보여준다.

목록 4-26. 표검색

```
public Vector getTables(String[] types) {
   Vector tableVector = new Vector();
   try {
      Connection  con  =  DriverManager.getConnection(url,  userName,
password);
      DatabaseMetaData dbmd = con.getMetaData();
      ResultSet rs = dbmd.getTables(null, null, "%", types);
```

```
        while (rs.next()) {
            tableVector.addElement(rs.getString("TABLE_NAME"));
        }
        con.close();
    } catch (SQLException e) {
        reportException(e.getMessage());
    }
    return tableVector;
}
```

표이름들을 얻는 코드는 표의 류형를 얻을 때의 코드와 비슷하다. 차이나는것은 getTables()메쏘드에 대한 인수목록에 있다. 메타자료메쏘드들을 사용할 때 이러한 류형의 인수들이 일반적이므로 이 인수들에 대하여 좀 더 보기로 하자.

getTables()메쏘드는 아래와 같은 4개의 인수들을 가진다.

```
getTables(String catalog,
          String schemaPattern,
          String tableNamePattern,
          String[] types);
```

- catalog — 2중인용부호쌍("")은 일람표가 없는 표들을 검색하며 null은 모든 표들을 검색한다.
- schemaPattern — 2중인용부호쌍("")은 구조도식이 없는 표들을 검색하며 null은 모든 표들을 검색한다.
- tableNamePattern — 이것은 SQL의 "LIKE"에서 리용되는 인수와 비슷한 표이름패턴이다. SQL통용문자가 적용된다.
- types — 포함하는 표류형의 배렬이다. null은 모든 류형을 돌려준다.

getTables()메쏘드는 일람표에서 쓸수 있는 표들에 대한 설명을 포함하고있는 ResultSet를 돌려준다. 결과모임은 표 4-2에서 보여주는 렬들을 포함한다.

표 4-2. getTables()가 돌려주는 렬

렬	렬이름	형	내 용
1	TABLE_CAT	String	표일람표(null일수도 있다.)
2	TABLE_SCHEM	String	표구조(null일수도 있다.)
3	TABLE_NAME	String	표이름
4	TABLE_TYPE	String	표류형: TABLE, VIEW, SYSTEM TABLE 등
5	REMARKS	String	표에 대한 설명문

주의: 어떤 자료기지들에서는 모든 표들에 대한 정보를 돌려주지 않을수도 있다.

DatabaseMetaData객체는 또한 getColumns()메쏘드를 리용하여 표에서 렬들에 대한 구체적인 정보를 검색하는데 리용할수도 있다.

getTables()메쏘드와 같이 getColumns()메쏘드도 ResultSet를 돌려준다.

```
public ResultSet getColumns(String catalog,
                            String schemaPattern,
                            String tableNamePattern,
                            String columnNamePattern);
```

- catalog　2중인용부호쌍("")은 일람표가 없는 표들을 검색하며 null은 모든 표들을 검색한다.
- schemaPattern　2중인용부호("")는 구조도식이 없는 표들을 검색하며 null은 모든 표들을 검색한다.
- tableNamePattern　이것은 SQL의 "LIKE"에서 리용되는 인수와 비슷한 표이름패턴이다. SQL통용문자가 적용된다.
- columnNamePattern　이것은 SQL의 "LIKE"에서 리용되는 인수와 비슷한 렬이름패턴이다. SQL통용문자가 적용된다.

목록 4-26의 코드에서 돌려주는 표정보를 리용하여 getColumns()메쏘드로 렬정보를 얻을수 있다. 목록 4-27에 실례를 보여준다.

목록 4-27. 렬자료 검색

```
public Vector getColumns(String tableName) {
   Vector columns = new Vector();
   Hashtable columnData;
   try {
      Connection con = DriverManager.getConnection(url, userName, pas
sword);
      DatabaseMetaData dbmd = con.getMetaData();
      String catalog = con.getCatalog();
      ResultSet rs = dbmd.getColumns(catalog, "%", tableName, "%");
      ResultSetMetaData md = rs.getMetaData();
      int nColumns = md.getColumnCount();
      String value;
      while (rs.next()) {
```

```
            columnData = new Hashtable();
            for (int i = 1; i <= nColumns; i++) {
                value = rs.getString(i);
                if (value == null) value = "<NULL>";
                columnData.put(md.getColumnLabel(i), value);
            }
            columns.addElement(columnData);
        }
        con.close();
    } catch (SQLException e) {
        reportException(e.getMessage());
    }
    return columns;
}
```

목록 4-25부터 4-27의 실례코드들은 DatabaseUtilities클라스에 추가하면 된다.

주의: 대부분의 DatabaseMetaData메쏘드들은 JDBC 2.0과 JDBC 3.0에서 추가 혹은 수정되였기때문에 사용자의 구동프로그람이 JDBC 2.0이나 JDBC 3.0에 적합하지 않는 경우 일부 DatabaseMetaData메쏘드들에 의해 SQLException가 던져질수 있다.

이제는 JTree에서 DatabaseMetaData를 현시해보자.

DatabaseMetaData객체로부터 검색된 표와 렬자료를 현시하는데 필요한 MetaDataFrame클라스는 JInternalFrame클라스를 확장한것이다. 이것은 목록 4-28에서 보여주는것처럼 JScrollPane에서 JTree를 현시하는데 리용된다.

목록 4-28. JTree에 DatabaseMetaData를 현시

```
package JavaDB_Bible.ch04.sec05;

import java.awt.BorderLayout;
import java.awt.Color;
import java.util.Hashtable;
import java.util.Vector;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

class MetaDataFrame extends JInternalFrame{
    protected JTree tree;
```

```
    protected JScrollPane JTreeScroller = new JScrollPane();
    protected DatabaseUtilities dbUtils;
    protected String dbName;
    protected String[] tableTypes;
    protected JPanel JTreePane1 = new JPanel();

    public MetaDataFrame(String dbName, DatabaseUtilities dbUtils) {
        setLocation(0, 0);
        setClosable(true);
        setMaximizable(true);
        setIconifiable(true);
        setResizable(true);
        getContentPane().setLayout(new BorderLayout());
        this.dbName = dbName;
        this.dbUtils = dbUtils;
        setTitle(dbName);
        init();
        setVisible(true);
    }

    // JInternalFrame의 초기화
    private void init() {
        JTreePane1.setLayout(new BorderLayout(0, 0));
        JTreePane1.setBackground(Color.white);
        JTreeScroller.setOpaque(true);
        JTreePane1.add(JTreeScroller, BorderLayout.CENTER);
        DefaultTreeModel treeModel = createTreeModel(dbName);

        tree = new JTree(treeModel);
        tree.setBorder(new EmptyBorder(5, 5, 5, 5));
        JTreeScroller.getViewport().add(tree);
        JTreePane1.setVisible(true);
        JTreeScroller.setVisible(true);
        tree.setRootVisible(true);
        tree.setVisible(true);
        getContentPane().add(JTreePane1, BorderLayout.CENTER);
    }

    // DefaultMutableTreeNodes를 리용한 TreeModel의 창조
    protected DefaultTreeModel createTreeModel(String dbName) {
      DefaultMutableTreeNode treeRoot = new DefaultMutableTreeNode(dbName);
        Vector tableTypes = dbUtils.getTableTypes();
```

```
        for (int i = 0; i < tableTypes.size(); i++) {
            DefaultMutableTreeNode tableTypeNode =
                new DefaultMutableTreeNode((String) tableTypes.elementAt(i));
            treeRoot.add(tableTypeNode);

          String[] type = new String[] {(String) tableTypes.elementAt(i)};
            Vector tables = dbUtils.getTables(type);

            for (int j = 0; j < tables.size(); j++) {
                DefaultMutableTreeNode tableNode = new
                       DefaultMutableTreeNode(tables.elementAt(j));
                tableTypeNode.add(tableNode);

            Vector columns = dbUtils.getColumns((String) tables.elementAt(j));
                for (int k = 0; k < columns.size(); k++) {
                 Hashtable columnData = (Hashtable) columns.elementAt(k);
                    DefaultMutableTreeNode columnNode =
                       new
DefaultMutableTreeNode(columnData.get("COLUMN_NAME"));
                    columnNode.add(new DefaultMutableTreeNode("TYPE_NAME: " +
                            columnData.get("TYPE_NAME")));
                    columnNode.add(new
DefaultMutableTreeNode("COLUMN_SIZE: " +
                            columnData.get("COLUMN_SIZE")));
                    columnNode.add(new
DefaultMutableTreeNode("IS_NULLABLE: " +
                            columnData.get("IS_NULLABLE")));
                    tableNode.add(columnNode);
                }
            }
        }
        return new DefaultTreeModel(treeRoot);
    }
}
```

목록 4-28을 보면 대부분의 작업이 createTreeModel()메쏘드에서 수행된다.

이 메쏘드는 먼저 표의 류형들에 대한 벡토르를 얻기 위하여 목록 4-25에서 보여준 getTableTypes()를 호출한다.

다음 매개 표류형에 대하여 그 류형의 표이름벡토르가 목록 4-26에서 보여준 getTables()메쏘드에 의해 돌려진다. 이것들은 매개 표들을 나타내는 표류형마디들에

소속된 DefaultMutableTreeNodes를 창조하는데 리용된다.

끝으로 매개 표에 대하여 렬서술자들에 대한 하쉬표(Hashtable)들의 벡토르가 목록 4-27에서 보여주는 getColumns()메쏘드호출에 의해 얻어진다. 이 정보는 그림 4-9에서 보여준 렬마디와 렬정보자식마디들을 창조하는데 리용된다. 여기서는 적은 량의 렬정보들만이 현시된다. getColumns()메쏘드가 제공하는 렬정보는 표 4-3에 보여주었다.

표 4-3. getColumns()가 제공하는 렬정보

렬이름	종류	의미
TABLE_CAT	String	표일람표(null일수도 있다.)
TABLE_SCHEM	String	표구조(null일수도 있다.)
TABLE_NAME	String	표이름
COLUMN_NAME	String	렬이름
DATA_TYPE	short	java.sql.Types에서의 SQL류형
TYPE_NAME	String	자료원천의존형이름
COLUMN_SIZE	int	렬크기
DECIMAL_DIGITS	int	분수형식의 수
NUM_PREC_RADIX	int	밑수(보통 10이나 2)
NULLABLE	int	• columnNoNulls - NULL값을 허용하지 말아야 한다. • columnNullable - NULL값을 명확히 허용. • columnNullableUnknown - NULL값을 인식못함.
REMARKS	String	해설문렬(null일수도 있다.)
COLUMN_DEF	String	기정값(null일수도 있다.)
CHAR_OCTET_LENGTH	int	렬에서 최대바이트수
ORDINAL_POSITION	int	표에서 렬의 첨수(1부터 시작)
IS_NULLABLE	String	• NO는 명확히 NULL로 하지 않는 렬을 의미 • YES는 렬이 NULL값을 허용할수 있다는것을 의미 • 빈 문자렬은 NULL값을 인식하지 못함

관계형자료기지관리체계기능에 대한 정보검색

자료기지의 구조를 서술하는것외에 DatabaseMetaData객체는 관계형자료기지관리체계그자체에 대한 일반정보에 접근하는 메쏘드들을 제공한다. 자료기지관리체계에 대하여 검색할수 있는 일부 정보들이 그림 4-10에 제시되였다.

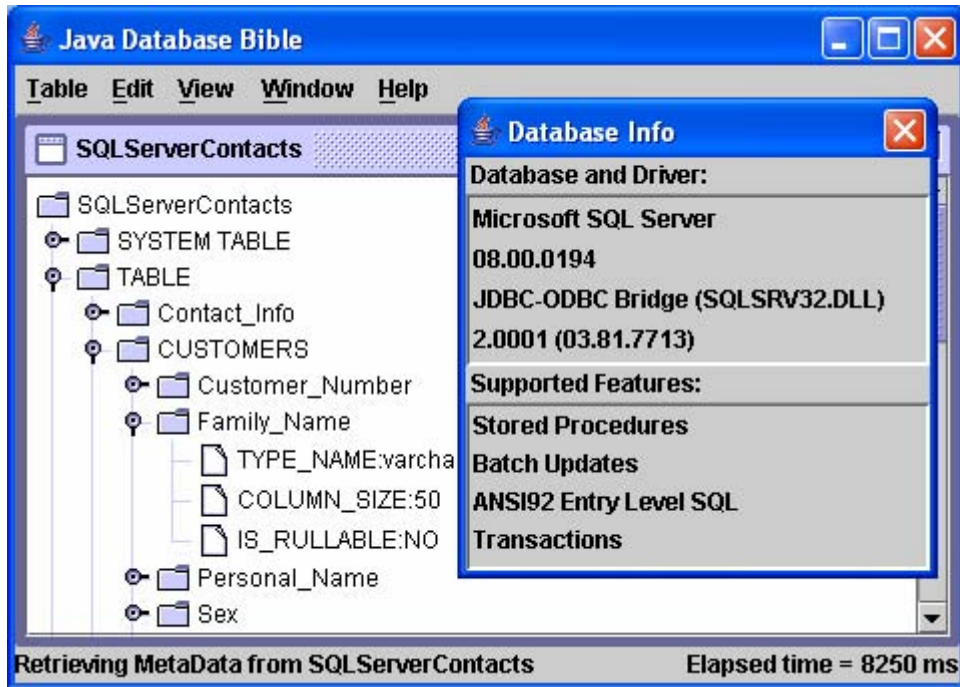



그림 4-10. 추가적인 DatabaseMetaData정보

그림 4-10의 실례는 SQLServerContacts자료기지가 JDBC-ODBC Bridge구동프로그람을 리용하면서 SQL Server에서 동작하고 있다는것을 보여준다. 또한 이 자료기지의 구성이 지원하는 일부 특징들을 렬거하였다.

상태띠에 보여주는 경과시간은 그림 4-9에 보여준 DatabaseMetaData의 나무보기에 접근하여 현시하는 시간이다. Opta2000구동프로그람을 사용하면 약 2s정도 걸리고 그림 4-10과 같이 Jdbc-Odbc Bridge를 사용하면 거의 9s정도 걸린다. 이 정보를 검색하는데 요구되는 코드를 목록 4-29에 보여주었다.

목록 4-29. 관계형자료기지관리체계에 대한 정보의 검색

```
package JavaDB_Bible.ch04.sec05;

import java.awt.*;
import java.util.Hashtable;
import java.util.Vector;
import javax.swing.*;
import javax.swing.JTree;
import javax.swing.border.*;
import javax.swing.tree.*;

public class InfoDialog extends JDialog{
   protected DatabaseUtilities dbUtils = null;
   protected JPanel dbInfoPanel = new JPanel();
   protected JPanel featuresPanel = new JPanel();
   protected JPanel otherPanel = new JPanel();
   protected JPanel topPanel = new JPanel(new BorderLayout());
   protected JPanel centerPanel = new JPanel(new BorderLayout());
   protected JPanel bottomPanel = new JPanel(new BorderLayout());

   public InfoDialog(DatabaseUtilities dbUtils) {
      this.dbUtils = dbUtils;
      setTitle("Database Info");
      getContentPane().setLayout(new BorderLayout());

      String[] dbInfo = dbUtils.databaseInfo();
      dbInfoPanel.setLayout(new GridLayout(dbInfo.length, 1, 2, 2));
      for (int i = 0; i < dbInfo.length; i++) {
         dbInfoPanel.add(new JLabel(dbInfo[i]));
      }

      dbInfoPanel.setBorder(new CompoundBorder(
            new BevelBorder(BevelBorder.LOWERED),
            new EmptyBorder(2, 2, 2, 2)));
      topPanel.add(new JLabel(" Database and Driver:"), BorderLayout.NORTH);
      topPanel.add(dbInfoPanel, BorderLayout.CENTER);
      getContentPane().add(topPanel, BorderLayout.NORTH);

      String[] features = dbUtils.featuresSupported();
      featuresPanel.setLayout(new GridLayout(features.length, 1, 2, 2));
      for (int i = 0; i < features.length; i++) {
         featuresPanel.add(new JLabel(features[i]));
      }

      featuresPanel.setBorder(new CompoundBorder(
```

```
                new BevelBorder(BevelBorder.LOWERED),
                new EmptyBorder(2, 2, 2, 2)));
        centerPanel.add(new JLabel("Supported Features:"), BorderLayout.NORTH);
        centerPanel.add(featuresPanel, BorderLayout.CENTER);
        getContentPane().add(centerPanel, BorderLayout.CENTER);
    }
}
```

보다싶이 이 실례에서는 DatabaseMetaData객체를 리용하여 얻을수 있는 자료의 일부만을 보여주고있다.

4.5.3. ResultSetMetaData

ResultSetMetaData객체는 ResultSet에 있는 렬들에 대한 특수한 정보를 돌려준다는것을 제외하면 DatabaseMetaData객체와 비슷하다.

결과모임속의 렬정보는 ResultSet의 getMetaData()를 호출하여 리용할수 있다. 돌려진 ResultSetMetaData객체는 그 ResultSet객체의 렬들의 개수와 자료형, 속성들을 준다.

표 4-4는 ResultSetMetaData객체의 메쏘드들중에서 보다 일반적인 일부 메쏘드들을 보여준다.

표 4-4. ResultMetaData메쏘드들

ResultSetMetaData메쏘드	설 명
getColumnCount()	ResultSet의 렬개수를 돌려준다.
getColumnDisplaySize(int column)	렬의 최대너비를 char형으로 돌려준다.
getColumnLabel(int column)	현시하는데 리용하는 렬제목을 돌려준다.
getColumnName(int column)	렬이름을 돌려준다.
getColumnType(int column)	렬의 SQL자료형색인을 돌려준다.
getColumnTypeName(int column)	렬의 SQL자료형의 이름을 돌려준다.
getPrecision(int column)	렬에서 10진수를 돌려준다.
getScale(int column)	소수점 아래자리수를 돌려준다.
getTableName(int column)	표이름을 돌려준다.
isAutoIncrement(int column)	만일 렬에 자동번호가 붙었다면 true를 돌려준다.
isCurrency(int column)	만일 렬값이 화페형값이면 ture를 돌려준다.
isNullable(int column)	만일 렬값이 NULL로 설정될수 있다면 ture를 돌려준다.

목록 4-30의 실례는 표의 렬이름과 렬개수를 모를 때 ResultSetMetaData메쏘드들인 getColumnCount()와 getColumnLabel()을 리용하는 방법을 설명하고있다.

목록 4-30. ResultSetMetaData를 리용

```
public void printResultSet(String query){
  try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection("jdbc:odbc:Inventory");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    ResultSetMetaData md = rs.getMetaData();

    int nColumns = md.getColumnCount();
    for(int i=l;i<=nColumns;i++){

System.out.print(md.getColumnLabel(i)+((i==nColumns)?"\n":"\t"));
    }
    while (rs.next()){
      for(int i=l;i<=nColumns;i++) {

System.out.print(rs.getString(i)+((i==nColumns)?"\n":"\t"));
      }
    }
  }
  catch(Exception e){
   e.printStackTrace();
  }
}
```

실례는 먼저 ResultSet에 대한 렬수를 검색하고 다음에 순환을 통하여 렬표식을 얻어 그것을 첫행에 써넣는다. 다음에 매 행을 순환하면서 내부순환으로 자료를 검색한다.

4장에 대한 요약

이장을 통하여 다음과 같은 문제들을 잘 리해해야 한다.

- 표구축자를 작성하기 위한 JDBC와 Swing의 리용
- COMMIT 및 ROLLBACK와 거래조종의 기초적인 리용
- ResultSetMetaData를 리용하여 표에 대한 정보얻기
- JDBC와 Swing을 리용한 JDBC/SQL표편집기 작성
- JDBC ResultSet와 ResultSetMetaData
- 실행효률을 개선하기 위한 색인의 작성과 리용
- DatabaseMetaData의 사용
- ResultSetMetaData의 사용

다음 장에서는 웨브응용프로그람 실례를 통하여 JDBC 2.0 확장 API를 배비한다.

제 5 장. JDBC 와 3 층 웨브싸이트

Java가 성공하게 된 기본요인중의 하나가 봉사기측 프로그람의 작성이다. Java프로그람들중에서 그중 널리 류포된것이 바로 써블레트, JSP 및 자료기지를 리용한 동적인 웨브싸이트작성이다.

제5장에서는 인터네트상에서 웨브응용프로그람개발을 실례로 JDBC확장 API를 취급한다. 이 웨브응용프로그람은 써블레트와 JSP로 업무론리(business logic)를 실현하는 Apache/Tomcat기초의 웨브봉사기를 중심으로 구축된 3층구성방식을 리용한다. 웨브봉사기는 자료기지봉사기와 접속하기 위하여 JDBC를 리용한다.

제1절. 회원웨브싸이트의 설계

자바와 자료기지를 함께 리용하는 분야가 바로 동적웨브싸이트를 만드는 분야이다. 이 절에서는 Apache/Tomcat에 기초한 웨브봉사기를 중심으로 구축된 3층구성방식을 리용하여 회원웨브싸이트를 설계한다.

5.1.1. 웨브싸이트의 기능

설계를 여러 층으로 나누어 진행하면 주어진 경우에 맞게 다양한 기성기술들을 적용할수 있다. 실례로 웨브층에 있는 JSP페지와 써블레트로부터 생성된 웨브페지를 현시하는 열람기가 의뢰기층을 조종한다. 이것은 사용자가 해야 할것은 HTTP명세를 지키기는것뿐이며 맞다들수 있는 모든 열람기들이 지원하지 않는 임의의 기술들을 피한다는것을 의미한다.

열람기와 관계형자료기지관리체계는 명백히 정의된 대면부를 가지고있는 기성제품들이기때문에 뒤따르는 절들에서 그것들과 대면하는데 필요한 업무론리와 자료표현론리에 집중한다. 열람기에 대한 대면은 정적인 웨브페지들을 봉사하며 Tomcat에 대한 앞단처리(front end)를 제공하는 웨브봉사기를 통하여 조종된다. Tomcat는 써블레트와 JSP페지들을 리용하여 Java로 작성된 동적인 웨브내용물(content)을 봉사한다.

그림 5-1에서는 3층체계의 구조를 보여주었다. 왼쪽이 표준웨브열람프로그람을 실행하고있는 의뢰기이고 가운데가 웨브봉사기, 오른쪽이 자료기지봉사기이다.

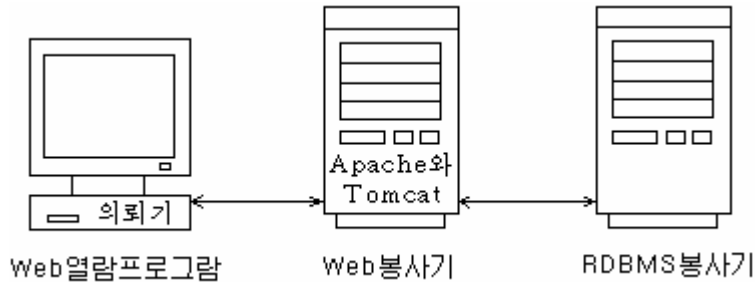



그림 5-1. 3층 구성방식의 웨브응용

업무론리와 자료표현론리는 웨브봉사기층에서 Java와 JSP를 리용하여 처리한다. 자료기지자체는 가상적으로 임의의 관계형자료기지관리체계를 리용할수 있다. 이 장의 실례들은 SQL Server와 Opta2000구동프로그람에 기초하고있다. Opta2000구동프로그람은 JDBC확장 API를 지원하는 순수한 Java구동프로그람이며 대부분의 자료기지들에서 다 리용할수 있다. 실례코드들에서 사용자이름, 통과암호, 봉사기이름을 변경하는것은 그리 힘들지 않으므로 서로 다른 구동프로그람들을 리용하며 각이한 관계형자료기지관리체계로 쉽게 절환할수 있을것이다.

실례들은 써블레트, JSP, JavaBeans에 중심을 두고 배비된다. 또한 실례들에서는 JDBC확장 API를 여러 각도에서 설명해준다.

웨브싸이트를 설계하는데서 첫 단계는 싸이트의 기능을 정의하고 밑준위에 놓이는 자료기지를 설계하는것이다. 자료기지가 지원하게 되는 웨브페지를 중심으로 자료기지를 설계하면 Java코드를 보다 간단하게 그리고 빨리 수행되도록 작성할수 있다.

웨브싸이트의 기능상요구

실례로 망상에서 서로 자기의 자동차를 소개하고 경험을 나누는 회원웨브싸이트를 구축하기로 하자. 웨브싸이트의 설계과정에 다음과 같은 JDBC의 주요 문제들을 고찰하게 된다.

- 써블레트, JSP, JDBC로 HTML폼 다루기
- 탐색엔진에서 흘리기가능한 ResultSet의 리용
- 회원이 자기의 인물자료를 불러내고 변경할수 있게 하는 갱신가능한 ResultSet의 리용
- HTML폼과 blob를 리용한 화상의 올리적재, 저장 및 검색조종
- 전자우편을 주고 받기 위한 JDBC와 JavaMail API의 리용

같은 ResultSet로부터 서로 다른 웨브페지들을 작성하기 위한 XML과 XSLT의 리용에 대해서는 갱신가능한 ResultSet를 다루면서 고찰한다.

실례로 보여주는 프로그람들은 일반적인 회원웨브싸이트의 표준적인 기능은 물론 대부분의 상업일람싸이트들에 공통적인 다음과 같은 기능들을 지원해준다.

- 회원 등록가입(log in)
- 새 회원 등록
- 회원자료기입
- 화상과 같은 대형객체들의 올리적재와 저장
- 싸이트탐색
- 작은 사진들을 포함하는 개요페지의 현시
- 개요페지로부터 상세페지들의 련결
- 자동화된 전자우편 지원

웨브싸이트의 론리적구조를 리해하는데서 가장 좋은 방도는 블로크도표를 리용하는것이다. 이 장에서 보게 되는 웨브싸이트의 론리적구조를 그림 5-2에 보여주었다.

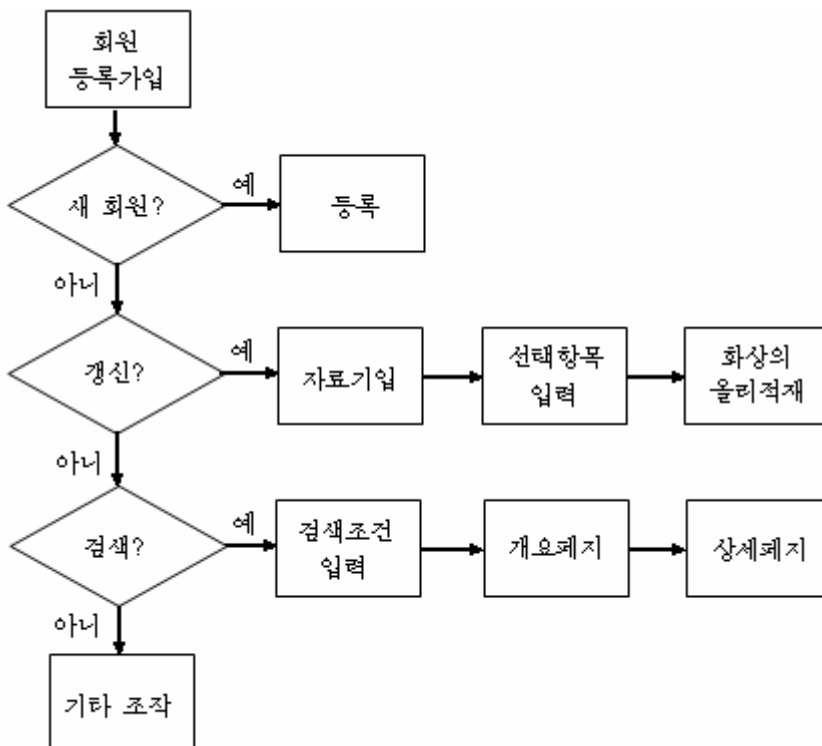



그림 5-2. 회원웨브싸이트의 구조

회원의 등록가입과 등록과정은 HTML폼들의 현시와 폼에 입력한 자료처리를 포함한다. 실례들에서는 론리기능들을 교갑화하기 위하여 JavaBeans를 리용하는 써블레트와 JSP방법들을 동시에 리용하고있다. 단순한 본문기초의 폼들에 대한 취급외에 열람기페지에서 화상들을 올리적재하고 그것들을 자료기지에 저장하는 방법을 보여준다.

5.1.2. 자료기지설계

회원등록가입처리

사용자는 먼저 사용자이름과 통과암호를 가지고 등록가입요청폼에 응답해야 한다. 등록가입시도에는 다음과 같은 3가지 가능한 결과들이 있을수 있다.

- 싸이트접근을 허락할수 있는 정확한 사용자이름과 통과암호에 의한 성공적인 등록가입
- 사용자이름은 정확하지만 통과암호가 틀려서 실패한 등록가입시도
- 사용자이름부터 정확치 않아 실패한 등록가입시도

사용자이름이 정확하지만 통과암호가 틀렸을 경우에는 전자우편을 암시해주는 통보문을 현시하며 사용자이름부터 완전히 틀린 시도는 사용자가 새로운 등록단계를 거치도록 안내한다.

등록가입표 자체는 아주 간단하다. 표 5-1에 보여주는 세개의 렬만을 리용한다. UserName렬이 기본열쇠이며 이 렬은 호출속도를 높이기 위하여 사전식으로 색인*된다. 사용자이름에 의한 접근이 빨라지는 대가로 새로운 사용자가 삽입될 때에는 색인의 재구축때문에 속도가 떠진다.

표 5-1. 등록가입표

UserName	Password	MemberID
ChoiYongSil	1979228	6
KimSongChol	tiger123	1
KimUnHui	bingbing	2
LiSongMin	hyne123	3
PakCholJin	pcj1000	5
RyoMiSong	snoopy	4

* 렬에 대한 색인은 그 렬이 문자렬형일 때에는 사전식으로 정렬되고 수자로 된 렬들일 때에는 커지는 순서대로 정렬된다.

Password렬은 단지 사용자의 정당성을 검사하는데 리용된다. 그러나 MemberID렬은 다른 표들에 접근하기 위한 외부열쇠이다. 대부분의 표들에서는 매개 회원이 오직 하나의 기입사항을 가지기때문에 MemberID가 기본열쇠로 된다.

회원등록

새 회원이 등록처리에 들어오면 체계는 그 회원에 대한 자료를 완성하는 HTML폼을 현시한다. 다음 폼에 입력한 자료가 표 5-2에 보여준 Contact_Info표에 보관된다.

표 5-2. Contact_Info표

ID	Family_Name	Personal_Name	State	City	Street	Phone	Email
1	김	성철	평양시	평양	창광거리	359-6532	tiger@home.com
2	김	은희	평양시	평양	천리마거리	546-2259	bing@cn.com
3	리	성민	평안북도	신의주	금수거리	123-456-1111	axman@abc.com
4	려	미성	함경남도	함흥	미래거리	059-399-1999	hana@sec.com
5	박	철진	평양시	평양	광복거리	386-5431	tree@pcj.com
6	최	영실	라선시	라선	부흥거리	049-258-1468	birthday@baby.com

Contact_Info표는 회원자료기지에서 회원들의 이름과 주소정보가 저장되여있는 유일한 곳이다. Contact_Info표의 기본열쇠는 MemberID(표에서는 ID로 표시함)이다.

등록폼을 완성한 다음에는 자동차정보를 기입하기 위한 추가선택항목이 주어진다. 이에 대한 자료는 기본적으로 Product_Info표와 Options표에 저장된다.

자료기입

자동차에 대한 자료는 자료기입의 편리성과 검색의 효과성을 위하여 많은 표들에 분할하여 보관한다. 기본표는 자동차의 제작회사(maker), 모형(model), 생산년도(year), 색갈(color)과 같은 자료들이 포함되는 Product_Info표이다. 부차적인 보조표들에는 선택적인 자동차의 부속물이나 사진과 같이 덜 중요한 자료들이 저장된다.

표 5-3에서 보여준 Product_Info표는 대부분의 탐색에 리용되며 따라서 효률적인 탐색을 담보하는것이 중요하다. 이것은 많은 렬들에 색인을 달게 된다는것을 의미한다.

이 표에서 기본열쇠는 VehicleID이다. 이 열쇠는 Product_Info표와 1대 1관계에 있는 Options표에서도 기본열쇠로 리용된다.

표 5-3. Product_Info표

VehicleID	MemberID	Maker	Body	Model	Year	Color
1000	1	Honda	Coupe	Civic	1996	Red
1001	1	Mitsubishi	SUV	Montero	2000	Green
1002	2	GM	Pickup	Sonoma	1999	Red

Product_Info표는 그림 5-3에서 보여준것과 같은 HTML폼을 리용하여 갱신된다. 일반적으로 자료기입폼에는 자료를 다른 장소에 기입하거나 탐색에 리용할수 없는 형식으로 입력하는 현상을 가능한껏 줄이기 위하여 복합칸(combobox)들을 많이 사용한다. 자유본문마당들은 탐색능력이 제공되지 않는데서만 사용한다.

Vehicle Description		
Body Style:	Select	(Convertible, Couple, SUV, ...)
Make:	Select	
Model:	Select	
Year:	Select	
Engine:	Select	(6 Cyllinder, Diesel, ...)
Transmission:	Select	(Automission, 6 Speed, ...)
Color:	Select	
Location:		(Your ZIP Code)
Reserve Price:		(Minimum acceptable price)
	Proceed to Options Page	

그림 5-3. 자료기입오유를 줄이기 위해 복합칸을 리용하는 자료기입폼

Product_Info표는 여러개의 개별적인 표들로 분해하는 관점에서 살펴볼 필요가 있다. 큰 표를 공통적인 탐색부류들로 구성된 보다 작은 여러개의 표들로 가르면 명백히 탐색속도가 빨라진다.

사진과 큰 블로크의 자유본문들을 저장하는 특수한 표들외에 나머지 표들은 서로 비슷하며 제품추가선택항목들과 같이 특유한 특징들을 식별하는 론리형변수들을 저장한다.

자료기지의 항목들이 많은 각이한 특징들을 가지는 보다 큰 프로그람에서는 자료기입을 간소화하기 위하여 많은 종류의 표들로 가르는것이 좋을것이다. 이런식으로 매개 표는 간단한 HTML폼으로 넘길수 있다.

매 표에 해당한 JSP페지들은 사용자가 다음 페지로 넘어가기전에 꼭같은 일반 SQL InsertBean을 리용하여 폼자료를 다룰수 있다.

SQL InsertBean은 http요청파라메터들을 반복하기 위하여 Enumeration대면부를 리용하며 SQL Insert문을 작성한다. 목록 5-1은 이러한 기술을 보여준다.

목록 5-1. 렬거형을 리용한 일반폼조종

```
// HTTP요청으로부터 이름과 값들을 얻기 위해 Enumeration을 리용
for(Enumeration e=request.getParameterNames();e.hasMoreElements();)
{
    pNames[i] = (String)e.nextElement();
    Values[i] = request.getParameter(pNames[i]);
    ++i;
}

// SQL의 INSERT지령을 위한 fieldNames문자렬과 fieldValues문자렬 작성
String fieldNames = "ID,";
String fieldValues = "'" + memberID + "',";

// fieldNames와 fieldValues에 파라메터이름과 값들을 추가한다.
for(int j=0;j<i;j++){
    if (!pNames[j].equals("DBName") &&
        !pNames[j].equals("TableName") &&
        !pNames[j].equals("SubmitButton")){
            fieldNames += pNames[j] + ",";
            fieldValues += "'" + fixApostrophes(Values[j]) + "',";
    }
}

// 마지막 반점을 제거한다.
fieldNames = fieldNames.substring(0,fieldNames.length()-1);
fieldValues = fieldValues.substring(0,fieldValues.length()-1);

// SQL지령을 작성한다.
SQLCommand = "INSERT INTO " + tableName + " (" +
              fieldNames + ") " + "VALUES (" + fieldValues + ")";
```

양식자료를 다루는 이런 일반적인 방법은 비록 표의 구조가 HTML양식들을 따르지만 중간층 코드는 표의 구조나 현시하는 양식과 독립일수 있다는것을 의미한다. 이러한 방법은 사용자가 새로운 표들을 추가하거나 이미 있는 표들을 갱신해야 할수 있으므로 유지보수를 훨씬 더 쉽게 해준다.

Product_Info표와 같이 대부분의 표들은 자료기입오유를 최소화할수 있게 설계된양식들로부터의 입력자료를 리용하여 완성된다. 이 경우 대부분의 기입사항들은 그림 5-4에서 보는바와 같이 검사칸들을 리용하여 만들어진다.

Select Options:		
☐ AM_FM_Radio	☐ Cassette	☐ Single CD
☐ Multi CD		
☐ Power Windows	☐ Power Locks	☐ Air Conditioning
☐ Rear Air Cond	☐ Tilt Steering	☐ Power Steering
☐ ABS	☐ Cruise Control	
☐ Overhead Console	☐ Extended Cab	☐ Sun Roof
☐ Moon Roof	☐ Alloy Wheels	☐ Roof Rack
☐ Tow Package	☐ Four Wheel Drive	
☐ Bench Seat	☐ Bucket Seats	☐ Power Seats
☐ Child Seat	☐ Leather Seats	
Other options:	[ ]	
	Click here to proceed	

그림 5-4. 검사칸들을 리용하여 만든 자료기입폼

그림에서 볼수 있는것처럼 단일한 자유형식의 본문마당은 검사칸들로 지원한다. 역시 이 방법의 근본리유는 자료기입의 오유를 최소화하는것이다. 검사칸들은 탐색을 빠르고 쉽게 하는 론리형변수들로 넘어간다.

표 5-4에서는 그림 5-4에 보여준 HTML양식을 리용하여 완성된 표의 단순화된 부분모임을 보여주었다. 표 5-4에서 제일 중요한 렬은 현시를 목적으로 표에서 항목들의 개요를 보여주는 List렬이다. 이 렬의 자료는 모든 자료기입마당들과 "Other options"마당의 내용들로부터 문자렬생성에 의해 그 표가 갱신될 때 하나로 종합된다.

표 5-4. Options표의 일부분

VehicleID	Tow_Bar	4WD	Other	List
1000	0	0		AM/FM Radio, Cassette, Moon roof, Power windows
1001	0	1	Entertainment center	AM/FM Radio, CD Changer, Moon roof
1002	1	0		AM/FM Radio, CD, Power locks, Power windows

지금까지 론의된 표들은 모두 탐색을 쉽게 하도록 구조화되였다. 쉬운 탐색을 위한 지원은 표들에 자료를 기입하는데 리용되는 HTML양식들을 통하여 실현되였다. 자료기지는 탐색할수 없는 다음과 같은 보충적인 표들도 포함한다.

- blob로 된 회원들의 사진을 저장하는 Photos표
- 자유형식의 본문을 저장하는 BodyText표

이 표들은 MemberID를 가지고 접근할수 있다. Photos표에서는 PhotoID가 기본열쇠이기때문에 MemberID를 외부열쇠로 리용한다.

Photos표는 사진들을 blob로 저장한다. 이 자료들에는 SQL용어로 위치자(locator)라고 하는 지적자를 리용하여 stream이나 byte배렬처럼 접근하므로 특별한 조종을 요구한다. 열람기에서 사진을 올리적재하는것도 써블레트객체가 제공하는 기본 HTML양식지원에 포함되지 않는 특별한 조종을 포함한다.

자료기지탐색

자료가 일단 자료기지에 저장되면 회원들은 탐색양식을 통하여 거기에 접근할수 있다. 자료기지에 대한 탐색은 그림 5-3의 자료기입양식과 류사한 탐색양식을 통하여 수행된다.(그림 5-5)

탐색결과는 페지마다 몇개의 자료기지항목개요들을 보여주는 개요양식으로 나타난다. 매개 개요항목들은 써블레트를 리용하여 자료기지에서 내리적재되는 작은 화상들을 포함한다. 개요페지의 일반적인 모양을 그림 5-6에 보여주었다.

그림 5-5. 자료기지에 대한 HTML탐색폼

그림 5-6. 자료기지에서 여러개 항목들의 개요를 보여주는 개요페지

개요페지에서 사용자는 작은 화상들을 찰칵하여 보다 구체적인 자료를 보여주는 페지를 선택할수 있다. 그림 5-7에 보여준것과 같은 상세페지는 실제적으로 XML로 검색되며 상세페지를 작성하는 봉사기상에서 XSL양식일람으로 처리된다.

그림 5-7. 보다 큰 화상과 보충적인 정보들을 보여주는 상세페지

상세페지를 창조하는 XML방법은 주로 갱신가능한 ResultSet의 리용을 설명하기 위하여 선정되였다. 갱신가능한 ResultSet는 먼저 보여준 소개자료처럼 현시되든지 미리적재되여 편집준비된 XML폼으로 현시된다. 같은 XML문서로부터 완전히 다른 현시형식을 생성하는것은 XML을 다른 HTML문서들로 전환하는 XSLT의 리용에 의해 가능해진다.

자료기지구동 전자우편

상세페지는 리용자가 자동차의 소유자에게 전자우편을 보낼수 있게 하는 본문령역을 가지고있다. 전자우편은 자료기지에서 송신자와 수신자정보를 얻고 통보문을 보내는 JavaMail프로그람을 통하여 처리된다. JDBC와 JavaMail을 결합하면 회원들에게 전자우편을 자동적으로 보내고 받을수도 있으며 그것들을 자료기지에 직접 보관할수 있다.

제2절. 써블레트, JSP의 리용

써블레트와 JSP는 Java기술의 위력을 봉사기측 응용프로그람에로 확장한것이다. 이것들은 CGI프로그람작성기술에 대한 Java기술의 대답이다. 왜냐하면 이것들을 리용하면 개발자들이 공동의 자료원천으로부터 얻은 정보와 사용자가 입력한 내용을 결합하는 동적인 웨브페지들을 구축할수 있기때문이다.

봉사기측 Java는 전통적인 Perl CGI에 비해 효과성 측면에서 의의있는 개선을 가져왔다. 전통적인 Perl CGI에서는 HTTP요청때마다 새로운 프로쎄스를 시작하기때문에 프로쎄스기동으로 인한 부가시간이 CGI프로그람의 실행시간보다 클수 있다. 써블레트와 JSP프로그람에서는 매개 요청이 무거운 조작체계 프로쎄스가 아니라 전기간 상주하고있는 Java가상기계의 가벼운 Java스레드에 의해 처리된다.

Java 써블레트와 JSP페지의 또 하나의 중요한 우점은 프로그람수가 전체 응용프로그람에서 단일한 개발언어를 리용할수 있게 한다는것이다. 다시말하여 Solaris가동기반에서 실행되는 Apache봉사기용 프로그람을 작성한다고 할 때 그 개발과 검사는 Java를 지원하는 임의의 조작체계에서 할수 있다.

이 절에서는 동적웨브페지를 작성하기 위하여 JSP를 리용하는 간단한 방법을 보여준다. 이 웨브페지들은 `DataSource`객체를 리용하여 접근되는 회원자료기지에 의해 구동된다.

5.2.1. 동적웨브페지작성을 위한 써블레트의 리용

써블레트(servlet)란 봉사기상의 써블레트엔진에서 실행되며 의뢰기의 요청을 받고 봉사하는 Java클라스를 말한다.

써블레트는 일반적으로 동적웨브페지작성에 많이 리용된다. 직결일람표(online catalog)가 바로 동적웨브응용프로그람의 전형적인 실례이다. 써블레트는 의뢰기로부터 요청을 받으면 자료기지에서 자료를 얻어내고 형식화하여 의뢰기에 돌려준다.

간단한 써블레트를 작성해보자.

모든 써블레트들은 `javax.servlet.Servlet`대면부를 실현하여 작성된다. 써블레

트들은 표준적으로 써블레트대면부를 실현하는 javax.servlet. GenericServlet를 확장하여 작성되든가 HTTP요청에 봉사하는 써블레트들의 기초클라스인 javax.servlet.http.HttpServlet를 확장하여 창조된다.

써블레트대면부는 기본생명주기과제들을 처리하기 위하여 써블레트엔진에 의해 호출되는 생명주기메쏘드들을 정의한다. 이 생명주기과제들은 초기화, 의뢰기요청봉사, 파괴, 페공간수집이다.

써블레트가 수행하는 대부분의 작업들은 의뢰기요청봉사메쏘드들에서 처리된다. 다음의 두가지가 HttpServlet클라스의 가장 중요한 의뢰기요청봉사메쏘드들이다.

- doGet: HTTP GET요청을 지원하기 위하여 재정의된다.
- doPost: HTTP POST요청을 지원하기 위하여 재정의된다.

GET와 POST는 봉사기에 요청파라메터들을 전송하는데 리용하는 CGI메쏘드들이다. GET요청에서는 파라메터들이 주콤퓨터의 URL에 추가되며 파라메터들의 최대길이가 256문자로 제한된다. 그러나 POST요청에서는 파라메터들이 별도로 넘겨지며 파라메터의 길이에 제한이 없으므로 보다 많은 자료를 보낼수 있다.

HTTP써블레트의 전형적인 리용에는 보통 다음과 같은것들이 포함된다.

- HTML폼이 제출하는 자료의 처리와 저장
- 동적웨브페지의 작성
- 직결물건사기와 같은 응용에서 상태정보관리

써블레트는 전통적인 CGI스크립트보다 많은 우월성을 가지고있으며 오늘날 응용프로그람봉사기의 골간이다. 목록 5-2의 간단한 실례를 통하여 알수 있는것처럼 써블레트를 작성하고 배비하는것은 그의 큰 능력에 비해 상대적으로 쉽다.

목록 5-2. 간단한 써블레트

```
package JavaDB_Bible.ch05.sec02;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
   protected void doGet(HttpServletRequest req,
                    HttpServletResponse resp) throws ServletException,
         IOException {
      resp.setContentType("text/html");
      PrintWriter out = resp.getWriter();
```

```
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello Servlet</TITLE></HEAD>");
        out.println("<BODY><H1>Hello Servlet World</H1></BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

이제는 우에서 본 실례를 확장하여 간단한 등록가입써블레트를 작성해보자. 회원웨브싸이트를 구성하는데서 제일 첫 단계는 회원등록가입을 처리하는것이다. 우리가 앞에서 설계한 웨브싸이트는 사용자이름과 통과암호가 들어있는 표를 호출한다.(표 5-5)

표 5-5. 사용자이름과 통과암호를 포함하고있는 등록가입표

UserName	Password	MemberID
KimSongChol	tiger123	1
KimUnHui	bingbing	2
LiSongMin	hyne123	3

이 표를 작성하기 위한 SQL지령은 다음과 같다.

```
CREATE TABLE LOGIN(
    UserName VARCHAR(20) PRIMARY KEY,
    Password VARCHAR(20) NOT NULL,
    MemberID int IDENTITY);
```

UserName렬이 기본열쇠로 정의되였지만 클라스터열쇠로 정의되지 않았다는데 대하여 주의해야 한다. 클라스터열쇠로 정의되는 경우 SQL Server자료기지의 물리적배치가 클라스터열쇠에 따라 정돈되기때문에 새 회원이 현재 범위안에 있는 클라스터열쇠값을 가지고 추가될 때 그 변화를 반영하기 위하여 표 전체가 갱신되여야 한다. 이것은 새로운 회원들이 많이 추가되는 경우 성능저하를 초래하게 된다.

사용자가 등록가입을 할 때 먼저 해야 할것은 등록가입표에서 그의 사용자이름과 통과암호를 찾는것이다. 이 탐색은 또한 필요한 임의의 다른 자료를 찾아볼수 있게 해주는 MemberID를 돌려준다. 표 5-6은 MemberID에 의해 색인될수 있는 회원이름과 주소표를 보여준다.

표 5-6. 회원이름과 주소표

ID	Family_ Name	Personal_ Name	State	City	Street	Phone	Email
1	김	성철	평양시	평양	창광거리	359-6532	tiger@home.com

회원과 자료기지사이의 사용자대면부는 HTML양식의 리용에 기초하고있다. 회원들은 간단한 HTML양식을 리용하여 웨브싸이트에 등록가입한다. 그림 5-8은 목록 5-3에 보여준 회원등록가입 HTML양식을 보여준다.

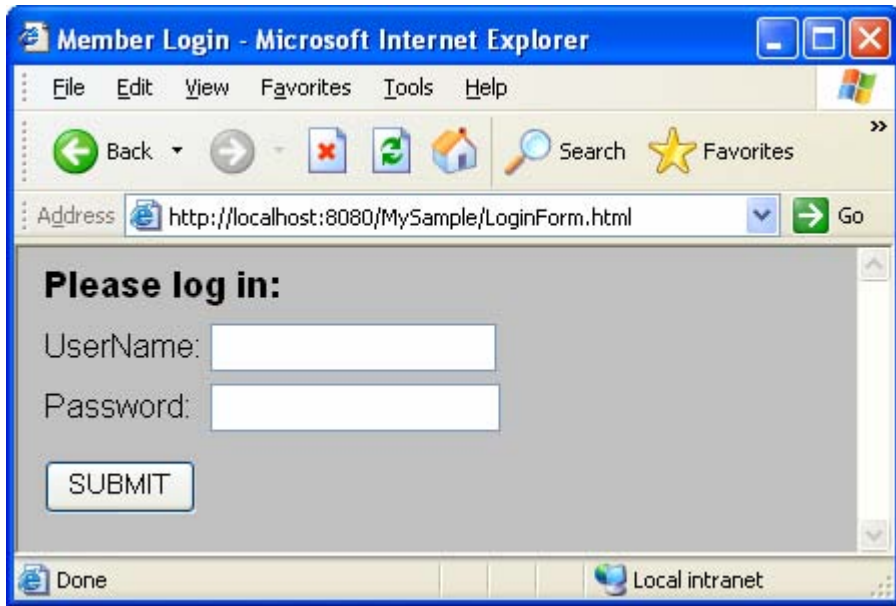



그림 5-8. 회원등록가입을 위한 HTML양식

목록 5-3. 기본등록가입양식(LoginForm.html)

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=big5">
   <title>Member Login</title>
</head>
<body bgcolor="#c0c0c0">
   <form method="POST"
         action="servlet/JavaDB_Bible.ch05.sec02.LoginServlet">
   <table>
      <tr>
         <td colspan=2>
         <h3>Please log in:</h3>
```

```
        </td>
      </tr>
      <tr>
        <td>Username:
        </td>
        <td>
        <input type="text" name="username"><br>
        </td>
      </tr>
      <tr>
        <td>
        Password:</td>
        <td>
        <input type="password" name="password"><br>
        </td>
      </tr>
      <tr>
        <td colspan=2> </td>
      </tr>
      <tr>
        <td>
        <input type="submit" value="SUBMIT" name="submitButton">
        </td>
      </tr>
    </table>
    </form>
</body>
</html>
```

목록 5-3에서 볼수 있는것처럼 action메쏘드는 POST메쏘드를 리용하여 JavaDB_Bible.ch05.sec02.LoginServlet를 호출한다. 그리고 두개의 입력마당 UserName과 Password가 파라메터로서 써블레트에 넘겨진다. GET메쏘드도 POST메쏘드와 꼭같이 동작할수 있으며 실제로 써블레트코드속에서 실현된다. POST는 GET메쏘드보다 융통성이 좋으므로 일반적으로 더 많이 쓰인다.

LoginServlet는 목록 5-2에서 보여주는 "Hello World"보다 그리 복잡하지 않다. 보다 실제적인 실례들에서 재정의되여야 할 기초적인 클라스메쏘드들은 다음과 같다.

- init()
- doPost()
- doGet()

목록 5-4의 코드는 JDBC구동프로그람을 적재하기 위한 init()메쏘드의 리용을 보여주고있다. doGet()와 doPost()메쏘드들은 사용자의 요청을 처리하기 위하여 재정의되여야 한다. writePage()메쏘드는 단순히 JDBC코드로부터 HTML출력을 분리하기 위한것이다.

목록 5-4. LoginServlet

```
package JavaDB_Bible.ch05.sec02;

import java.io.*;
import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   private Connection con = null;
   private DataSource ds = null;

   public void init(ServletConfig config) throws ServletException {
      super.init(config);
      try {
         Class.forName("com.inet.pool.PoolDriver");
         com.inet.tds.TdsDataSource tds =
               new com.inet.tds.TdsDataSource();
         tds.setServerName("DOLPHIN");
         tds.setDatabaseName("MEMBERS");
         tds.setUser(dbUserName);
         tds.setPassword(dbPassword);
         ds = tds;
      } catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }

   public void doGet(HttpServletRequest request,
                  HttpServletResponse response) throws
         ServletException, IOException {
      doPost(request, response);
   }

   public void doPost(HttpServletRequest request,
```

```
                   HttpServletResponse response) throws
          ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = new PrintWriter(response.getWriter());

      int id = -1;
      String memberPwd = null;
      String userName = request.getParameter("username");
      String password = request.getParameter("password");

      try {
          Connection con = ds.getConnection(dbUserName, dbPassword);
          Statement stmt;

          ResultSet rs = null;
          String SQLQuery = "SELECT * FROM LOGIN WHERE UserName = '" +
                         userName + "';";
          stmt = con.createStatement();
          rs = stmt.executeQuery(SQLQuery);

          while (rs.next()) {
              memberPwd = rs.getString("Password");
              id = rs.getInt("MemberID");
          }
          con.close();
      } catch (SQLException e) {
          System.err.println(e.getMessage());
      }

      RequestDispatcher dispatcher = null;
      if (id == -1) {
          dispatcher =
                getServletContext().getRequestDispatcher(
                "/jdbc/NewMember.html");
      } else if (!memberPwd.equals(password)) {
          dispatcher =
                getServletContext().getRequestDispatcher(
                "/jdbc/BadPassword.html");
      } else {
          dispatcher =
                getServletContext().getRequestDispatcher(
                "/jdbc/WelcomeBack.html");
      }
      dispatcher.forward(request, response);
  }
}
```

LoginServlet는 HttpServletRequest객체로부터 사용자가 입력한 UserName과 Password를 얻고 UserName을 SQL문에 밀어넣는다. 질문은 등록가입표로부터 일치하는 행들을 돌려준다. 사용자에게 적합한 응답을 생성하기 위하여 돌아온 값들에 대한 일련의 간단한 검사가 진행된다.

JDBC에 대한 초기화는 init()메쏘드에서 수행된다. 이 실례에서는 Opta2000구동프로그람을 리용하고있는데 대응하는 문자렬변수에 적당한 구동프로그람의 이름과 URL을 대입하여 임의의 구동프로그람으로 교체할수 있다.

DriverManager로부터 Connection을 얻기 위해 JDBC 핵심 API를 리용하는 고전적인 방법대신 이 써블레트는 javax.sql.DataSource를 리용하는 보다 좋은 접속방법을 보여준다. DataSource대면부를 리용하여 얻는 자료기지접속은 DriverManager가 제공하는 기본 Connection객체들보다 많은 능력 즉 접속공용과 분산거래를 지원할수 있다. 물론 실례에서 DriverManager를 대신 리용할수도 있다.

doPost()메쏘드외에 doGet()메쏘드가 doPost()에 대한 단순한 호출을 포함하고 있다는데 대하여 류의해야 한다. 이렇게 하는 리유는 전체 GET문자렬을 열람기의 주소창에 간단히 입력하는것에 의해 열람기로부터 써블레트의 조사를 쉽게 하기 위해서이다. 아래에 이에 대한 실례를 보여준다.

```
http://localhost:8080/servlet/JavaDB_Bible.ch05.sec02.LoginServlet?us
ername= OneFish&&password=TwoFish
```

SQL질문이 돌려주는 ResultSet는 MemberID가 이 사용자이름에 할당되였는가를 결정하는데 리용된다. 만일 할당되지 않았다면 써블레트는 사용자를 새회원가입서명페지(NewMember.html)로 안내한다. 류사하게 사용자이름은 인식되였지만 통과암호가 틀릴때에는 사용자를 재시도 혹은 통과암호를 전자우편으로 보낼수 있게 하는 페지(BadPassword.html)로 인도한다. 만일 자료기지에서 사용자이름과 통과암호가 발견되면 사용자는 Welcome페지(WelcomeBack.html)로 안내된다.

적당한 페지에로의 이동은 javax.servlet.RequestDispatcher객체를 리용하여 처리된다. RequestDispatcher는 요구하는 URL을 인수로 넘겨주는 ServletContext객체의 getRequestDispatcher()메쏘드를 리용하여 얻는다. URL을 지적하는 인수의 형식은 사선("/")뒤에 상대경로를 주고 마지막에 자원의 이름을 쓴다. 써블레트를 호출하자마자 사용자의 환경은 HTTP봉사기환경(Apache밑에 있는 가상주콤퓨터의 뿌리등록부일수 있다)으로부터 Tomcat의 뿌리등록부에 있는 써블레트환경으로 이행되기때문에 이것을 잘 기억해두는것이 중요하다.

써블레트의 일감이 수행되였을 때 그리고 다음 페지가 써블레트의 기능들로부터 또

하나의 자원이 그것을 처리할수 있는 그러한 범위까지 론리적으로 분리되였을 때 다음 페지에로의 이동을 리용해야 한다.

그러나 만일 개발과정에 ServletOutputStream이나 PrintWriter를 리용하여 써블레트로부터 나오는 임의의 출력을 이미 서술했다면 RequestDispatcher.forward메쏘드를 리용할수 없다. 그렇게 되면 IllegalStateException오유가 발생한다. 이 경우에는 대신 RequestDispatcher.include()메쏘드를 리용할수 있다.

등록가입써블레트를 배비하려면 적당한 등록부에 그 클라스파일을 넣어야 한다. 단순한 Tomcat설치에 대한 보통의 경로는 다음과 같다.

```
TOMCAT/WEBAPPS/ROOT/WEB-INF/CLASSES
```

Tomcat는 열람기의 주소창에 입력하는 /servlet/경로가 이 등록부와 대응되도록 URL넘기기를 정의하는 구성파일을 가지고있다. 사용자는 이 파일을 편집하여 요구하는대로 자기의 대응관계를 설정할수 있다.

실례에서 Opta-xs구동프로그람을 리용하고있는데 이것을 리용하려면 적당한 등록부에 Opta.jar를 놓고 Tomcat/conf등록부의 tomcat.properties파일에서 Tomcat의 클라스경로를 변경한다.

실례로 .jar파일이 /lib등록부에 저장되여있다면 tomcat.properties파일에 다음의 행을 추가하여 Tomcat의 클라스경로를 변경한다.

```
Wrapper.classpath=lib/Opta.jar
```

Tomcat판본 4.0부터는 tomcat.properties파일이 없으므로 클라스경로를 체계환경변수 CLASSPATH에 추가해주어야 한다.

5.2.2. JSP의 리용

목록 5-4에서 본바와 같이 JDBC를 리용하여 Java봉사기측 프로그람을 작성하고 배비하는것은 생각보다 힘들지 않다. Java봉사기측 프로그람을 작성하는데는 자바써버페지(Java Server Page: JSP)를 리용하는 보다 쉬운 방법이 있다.

JSP는 HTML페지안에서 Java코드를 리용하기 위한 수단을 제공한다. HTML페지의 정적인 부분은 보통의 방법으로 간단히 쓰고 특수한 태그안에 Java코드를 매몰한다. 이때 Java코드부분은 JSP엔진에서 실행되고 그 결과는 HTML로 의뢰기에 전송된다.

JSP를 구축하기 위한 4가지 주요 요소들은 다음과 같다.

- 표식언어요소. Web페지의 경우 그것은 HTML이다.
- Java코드블로크를 지적할수 있는 스크립트요소

- JSP구조와 환경을 조종하는 JSP지령문들
- 사용자가 파라메터적재와 같은 집행할 지령들을 지적하는 동작(action)

JSP특정의 요소들을 식별하는 태그들은 보통의 표식언어태그들과 혼돈하지 않도록 특수한 형태를 리용한다. 태그들은 다음 두가지 형식들중 어느 하나로 설정한다.

- `<%    %>`
- `<jsp:   />`

JSP가 써블레트에 비해 많은 우점들을 가지고있지만 실제로는 써블레트기술우에서 구축된다. JSP페지가 처음으로 요청될 때 JSP엔진은 JSP를 써블레트로 변역한다. 이 번역으로 인한 시간지연을 첫 최종사용자가 느끼지 못하도록 하기 위한 여러가지 번역방법이 있다.

JSP의 우월성을 론증하기 위해 JSP를 리용하여 등록가입실례를 다시 만들어보기로 하자. JSP엔진이 동적 HTML처럼 쉽게 정적 HTML을 봉사할수 있으므로 등록가입폼을 JSP페지로 돌려놓을수 있다. 목록 5-5에서 보여준 등록가입폼은 목록 5-3에서 보여준것과 기본적으로 같은 폼이다. 등록가입폼을 JSP페지로 만들기 위해 필요한것은 JSP엔진이 찾을수 있는 위치에 그것을 보관하고 LoginForm.jsp로 이름을 다는것이다.

목록 5-5. JSP를 리용한 등록가입폼(LoginForm.jsp)

```
<html>
  <head>
    <title>Member Login</title>
  </head>
  <body bgcolor="#c0c0c0">
    <%@ page language="java" contentType="text/html; charset=big5"%>
    <form method="POST" action="ProcessLogin.jsp">
      <table>
        <tr>
          <td colspan=2><h3>Please log in:</h3></td>
        </tr>
        <tr>
          <td>Username:</td>
          <td><input type="text" name="username"></td>
        </tr>
        <tr>
          <td>Password:</td>
          <td><input type="password" name="password"></td>
        </tr>
        <tr>
          <td colspan=2></td>
```

```
      </tr>
      <tr>
        <td></td>
        <td><input type="submit" value="SUBMIT" name="submitButton"></td>
      </tr>
    </table>
  </form>
 </body>
</html>
```

Login.html과 LoginForm.jsp의 기본차이는 action파라메터가 ProcessLogin.jsp를 가리키도록 변경한것뿐이다. UserName과 Password파라메터는 LoginServlet에서와 꼭같이 action메쏘드에 넘겨진다.

목록 5-6에서는 HTTP요청파라메터들을 받고 의뢰기에 대답하는 간단한 JSP페지를 보여준다. 이 실례는 HTML과 몇개의 JSP특정태그들을 리용하여 단일한 JSP페지에 매몰되는 Java를 결합하는 방법을 보여준다.

- <%　　%> : Java스크립트릿트들을 직결하기 위한 구분기호
- <%=expression %> : 식의 평가값을 계산하여 출력
- <%@ page %> : 페지속성을 정의하는 지령문

목록 5-6. CGI파라메터들을 현시하기 위한 JSP페지의 리용(ProcessLogin.jsp)

```
<HTML>
 <HEAD>
  <TITLE>
   Display Login Parameters
  </TITLE>
 </HEAD>
 <BODY>
  <p><h3>Your username and password are as follows.</h3>
   <%@ page language="java" contentType="text/html; charset=big5"%>
  </p>
  <p>
   <%
   String userName=request.getParameter("username");
   String password=request.getParameter("password");
   %>
   Username = <%=userName%></p>
   <p/>
   Password = <%=password%><p/>
 </BODY>
</HTML>
```

이 실례는 목록 5-4의 JDBC써블레트실례에서 리용한 SQL질문코드들을 포함시켜 확장할수 있다. 다시말하여 본래 써블레트실례의 방식을 모방하여 JDBC코드와 HTML생성을 둘다 결합할수 있다. 그러나 이와 같이 Java와 HTML을 한개 페지에 결합하여 JSP페지를 만드는것은 그리 좋은 방법이 못된다.

JSP페지를 구축하는 보다 좋은 방법은 MVC구조에서 모형으로 동작하는 JavaBean에 JDBC론리를 교갑화하는것이다. 뷰는 `<jsp:forward />`지령문을 리용하여 JSP페지에 의해 제공된다. 그러면 LoginForm.jsp의 `action`메쏘드가 bean을 적재하는 JSP조종기를 호출하고 요청파라메터를 넘겨주며 JavaBean이 실행하는 SQL의 결과에 따라 사용자를 적절한 JSP뷰에로 안내한다.

등록가입양식을 처리하는 MVC방법을 실현하기전에 JSP페지와 JavaBeans의 리용을 살펴보자.

5.2.3. JSP와 JavaBean의 리용

JSP의 가장 쓸모있는 기능들중의 하나는 JSP가 `<jsp:useBean>`태그를 써서 JavaBean의 리용을 직접 지원하는것이다.

JavaBean들은 이름에 의해 적재될수 있으며 그밖의 특정한 규칙들을 만족시키는 Java클라스들이다. 이 규칙들에는 다음과 같은것들이 포함된다.

- 공개클라스여야 한다: public class `JavaBean`
- 공개이면서 인수가 없는 구축자를 가져야 한다: public `JavaBean()`
- 비공개자료마당만을 리용해야 한다: private `String` message
- 자기의 비공개자료마당들에 대한 공개 접근자메쏘드들을 제공해야 한다:
    - public `getMessage()`
    - public `setMessage(String message)`
- 자기관찰기능 즉 자기의 행동에 대하여 bean에 질문하는 외부클라스의 능력을 지원해야 한다.

JSP프로그람에서 JavaBeans리용의 가장 큰 우점은 JSP페지의 론리를 별도의 Java클라스로 실현하고 그것들을 조립가능한 부분품으로 만든다는것이다. 이 방법은 다음과 같은 중요한 우점들을 제공한다.

- 론리로부터 내용을 분리
- 공통적인 과제들에 재사용가능한 끼움식 부품(구성요소)

JSP와 함께 리용될 때 JavaBeans는 론리를 현시로부터 분리하는것을 기본목적으로 하는 론리블로크로서의 기능과 자료저장을 기본목적으로 하는 저장클라스로서의 기능을 가진다.

이름에 의한 적재 : <jsp:useBean>태그

이름에 의해 JavaBean을 적재하고 실행하는 능력이 바로 JavaBeans를 끼움가능한 부분품들로 리용하기 위한 실제적인 열쇠이다. JavaBean은 콤파일할 때가 아니라 실행될 때 JSP와 련결된다. 그러므로 JSP는 JavaBeans의 업무론리와 별도로 편집하고 갱신될 수 있다.

JSP로부터 JavaBeans를 호출하기 위해서는 다음과 같이 쓴다.

```
<jsp:useBean id="TestBean" class="JavaDB_Bible.TestBean"/>
```

<jsp:useBean>요소는 많은 속성들을 가진다. 그중에서 가장 일반적으로 리용되는 것들은 다음과 같다.

- id="beanInstanceName": id속성은 JSP페지에서의 참조를 위하여 JavaBean에 국부이름을 할당한다. id는 대소문자를 구별하며 bean의 전체 범위에서 일관해야 한다.
- scope="page | request | session | application": scope속성은 bean이 존재하며 id에서 명명된 변수가 리용될수 있는 범위를 정의한다. 기정값은 page이다.
- class="package.class": class속성은 지적된 id를 가진 JavaBean이 이미 정의된 범위에 적재되지 않았을 때 적재하는 JavaBean클라스를 규정한다.

주의: <jsp:useBean>태그는 먼저 지적된 이름을 가진 bean구체례를 찾으며 지적된 범위내에서 bean구체례를 찾을수 없을 때에만 새로운 구체례를 만든다. 만일 bean이 다른 <jsp:useBean>요소에 의하여 이미 창조되였다면 id값은 본래의 <jsp:useBean>요소에서 리용된 id의 값과 일치해야 한다.

Scope

JavaBean이 일단 적재되면 자기의 범위에 따라 프로그람의 여러곳에서 호출될수 있다. 다시말하여 JavaBean의 범위(scope)는 bean을 호출할수 있는 프로그람의 부분을 정의한다. 기정값은 page이다. 여러 범위들의 의미는 다음과 같다.

- page – 그 페지가 의뢰기에 응답을 보내거나 다른 파일에 요청을 넘길 때까지 bean은 <jsp:useBean>요소가 있는 JSP페지 혹은 그 페지의 임의의 정적포함파일들로부터 접근될수 있다.

- request - bean은 JSP페지가 의뢰기에 응답을 보내거나 다른 파일에 요청을 넘길 때까지 같은 요청을 처리하는 임의의 JSP페지로부터 접근될수 있다.
- session - bean은 bean을 창조한 JSP페지와 같은 대화접속안에 있는 임의의 JSP페지로부터 호출될수 있다. bean은 전체 쎄션에 걸쳐 존재하며 그 쎄션에 분할되여있는 임의의 페지에서 리용할수 있다. 사용자가 bean을 창조하는 페지는 session=true와 함께 <%@ page %>명령문을 포함해야 한다.
- application - bean은 bean을 창조한 JSP페지와 같은 응용프로그람에 있는 임의의 JSP페지로부터 호출될수 있다. 이때 bean은 전체 JSP응용프로그람에 걸쳐 존재하며 그 프로그람에 속한 임의의 페지가 그 bean을 리용할수 있다.

경고: <jsp:useBean>태그를 리용할 때 이 태그의 끝에 닫는 기호 "/"를 꼭 쓰는것이 매우 중요하다. "/"을 쓰지 않으면 예측할수 없는 결과가 나올수 있다.

속성: <jsp:getProperty>태그와 <jsp:setProperty>태그

JavaBean의 속성들은 미리 정의된 접근자메쏘드나 취득자 및 설정자메쏘드들을 통하여 접근할수 있는 비공개자료마당이다. 취득자메쏘드와 설정자메쏘드의 이름들은 설계패턴이라고 부르는 특정한 규칙에 따른다. 이러한 설계패턴에 기초한 메쏘드의 이름들을 사용하여 JSP페지들은 JavaBean의 속성들에 접근할수 있다.

```
<jsp:setProperty name="TestBean" property="service"
   value= "login"/>
<jsp:setProperty name="TestBean" property="username"
   value=<%=userName%>/>
<jsp:setProperty name="TestBean" property="username"
   value='<%=request.getParameter("username")%>'/>
<jsp:getproperty name="TestBean" property="message"/>
```

JavaBean의 파라메터값을 설정하는 방법에는 여러가지가 있다. 첫번째 실례는 정적값을 리용하여 파라메터를 설정하고있다. 두번째 실례는 값을 설정하기 위한 평가식의 리용을 보여준다. 세번째 경우에 파라메터값은 암시적으로 CGI질문파라메터들에 있는 같은 이름의 값으로 설정된다. 네번째 방안은 뒤에서 자기관찰이라는 소제목하에 론의한다.

초기화를 위한 <jsp:setProperty>의 리용

JSP엔진은 범위내에서 bean구체레를 찾을수 없는 경우에만 그의 새 구체레를 적재한다. 이것은 bean을 자기 범위내에서 응용프로그람의 자료를 기록해두는 저장용기로 리용할수 있다는것을 의미한다.

그러면 bean을 어떻게 초기화하는가 하는 물음이 제기된다. 그 대답은 아래에 보여주는바와 같이 <jsp:useBean>요소자체내에 초기화부분을 앉히는것이다.

```
<jsp:useBean id= "TestBean" class="JavaDB_Bible.ch05.sec02.TestBean"/>
   <jsp:setProperty        name="TestBean"        property="message"
value="goodbye"/>
</jsp:useBean>
```

<jsp:useBean>요소내에 들어있는 임의의 <jsp:setProperty>요소들은 bean이 처음으로 적재되여 달릴 때에만 집행된다. 만일 현재 범위내에 bean이 이미 존재한다면 이 요소들은 실행되지 않는다. 즉 bean이 리용되는 첫번에만 그것이 초기화되고 뒤따르는 참조들은 초기참조에 의해 bean에 저장된 자료를 요구한다.

자기관찰

자기관찰(introspection)이란 JavaBean내부를 조사하여 사용자에게 쓸모있는 메쏘드들을 확인하는 능력을 말한다. 실례로 자기관찰은 JSP엔진이 JSP페지에서 설정할수 있는 JavaBean의 속성들을 볼수 있게 한다. 이것은 아래에서 보는바와 같이 JSP엔진이 속성설정의 대체적인 부분들을 자동적으로 조종할수 있다는것을 의미한다.

```
<jsp:setProperty name="TestBean" property="*"/>
```

<jsp:setProperty>태그에서 속성값으로 통용문자 "*"를 리용하면 JSP엔진이 JavaBean의 속성들을 식별하기 위해 자기관찰을 리용하며 속성들을 폼이 넘겨준 파라메터들로 설정한다는것을 의미한다. 목록 5-7에서 이 방법을 설명하였다.

목록 5-7. <jsp:useBean>태그와 함께 JSP리용(TestBean.jsp)

```
<html>
  <head>
    <title>ParameterTestBean</title>
  </head>
  <body>
    <%@ page language="java"%>
    <jsp:useBean id="ParameterTestBean" scope="request"
              class="JavaDB_Bible.ch05.sec02.ParameterTestBean"/>
    <jsp:setProperty name="ParameterTestBean" property="*"/>
    User Name:
    <jsp:getProperty name="ParameterTestBean" property="username"/><p/>
    Password:
    <jsp:getProperty name="ParameterTestBean" property="password"/>
  </body>
</html>
```

물론 파라메터이름들과 bean속성들사이에 1대 1대응이 없다면 항상 그것들을 수동적으로 맞추어야 한다. 덧붙여 말한다면 이 기술은 라지오단추의 값들을 보관하는데도 리용되는데 이때 선택된 라지오단추값에 따라 JavaBean의 속성이 설정된다.

목록 5-8의 간단한 JavaBean을 리용하여 JSP와 JavaBean의 사용을 검사할수 있다. 이것은 목록 5-7의 JSP코드와 함께 처음으로 리용되는 JavaBean이다.

목록 5-8. getter와 setter메쏘드를 설명하는 간단한 JavaBean

```
package JavaDB_Bible.ch05.sec02;

public class ParameterTestBean extends java.lang.Object{
   protected String username;
   protected String password;

   public ParameterTestBean(){
   }

   public void setUsername(String username){
      this.username = username;
   }

   public void setPassword(String password){
      this.password = password;
   }

   public String getUsername(){
      return username;
   }

   public String getPassword(){
      return password;
   }
}
```

같은 이름을 가지는 여러개의 검사칸들이 서로 다른 값을 지적한다면 사용자는 간단히 JavaBean의 속성을 String배렬로 지적한다. 유감스럽게도 String배렬의 값들은 쉽게 얻을수 없다. 이 경우에는 아래에 보여준것처럼 스크립트릿트를 리용해야 한다.

```
<%
  String[] checkBoxes = formHandler.getCheckBoxes();
  for(int i=0; i<checkBoxes.length; i++){
     if(i>0) out.print(",");
     out.print( " " + checkBoxes[i] );
  }
%>
```

JavaBean에서 내장 JSP객체들의 리용

JSP API는 내장된 잠재적인 객체들의 모임을 통하여 의뢰기와 JSP의 전후관계(context)에 대한 가치있는 정보령역에로의 접근을 제공한다. javax.servlet.jsp.PageContext객체는 대부분의 내장된 JSP객체들을 위한 일반 접근점이다. 내장된 JSP객체들은 다음과 같다.

- request
- response
- out
- session
- application
- pageContext
- page
- exception

request객체는 열람기에서 들어온 현재의 요청을 교갑화한다. 써블레트포함기는 ServletRequest객체를 창조하고 그것을 써블레트의 봉사메쏘드에 넘겨준다. ServletRequest객체는 파라메터의 이름과 값, 속성, 입력흐름과 같은 자료를 제공한다. 리용할수 있는 request객체의 메쏘드들은 다음과 같다.

- getQueryString()
- getHeader(String headerName)
- getCookies()

response객체는 의뢰기에 응답을 보내는데서 써블레트를 돕기 위한것이다. 써블레트포함기는 ServletResponse객체를 창조하고 자기의 봉사메쏘드에 그것을 넘겨준다. ServletResponse객체는 내용물의 류형과 같은 응답파라메터들을 설정할수 있게 해준다. 또한 사용자는 response객체로부터 2진출력을 위해 OutputStream을 얻을수 있다.

out객체는 의뢰기의 열람기에 출력하는데 리용되는 JspWriter의 구체례이다. JspWriter에 직접 접근하면 사용자는 스크립트릿트로부터 출력내용을 직접 보낼수 있다.

session객체는 HttpSession의 구체례이다. 이것은 쎄션범위내에서 bean이나 JSP페지에 의하여 읽고 쓸수 있는 객체형식으로 쎄션정보를 교갑화한다.

application객체는 ServletContext객체의 구체례이다.

pageContext객체는 대부분의 내장객체들에 대한 일반 접근점이다. 즉 실례로 쎄션객체를 얻자면 다음과 같이 호출할수 있다.

```
HttpSession session = pageContext.getSession();
```

page객체는 현재페지에 대한 참조이다.

exception객체는 현시되여야 할 오유페지를 일으키는 례외에 접근하기 위하여 오유페지에 의해 리용된다.

자동형변환

JavaBean을 작성할 때 integer나 double같은 여러가지 형의 속성변수들을 리용할 수 있다. 그런데 의뢰기에서 봉사기로 보내는 요청파라메터들의 값은 언제나 String형이다. 이 값들은 적당한 valueOf(String)식에 의해 자동적으로 다른 자료형으로 변환된다.

```
valueOf(String)
```

JSP의 자동형변환은 표 5-7에서 보여 준 메쏘드들을 리용한다.

표 5-7. JSP에 의해 제공되는 자동형변환메쏘드들

자 료 형	변 환 메 쏘 드
boolean 혹은 Boolean	java.lang.Boolean.valueOf(String)
byte 혹은 Byte	java.lang.Byte.valueOf(String)
char 혹은 Character	java.lang.Character.valueOf(String)
int 혹은 Integer	java.lang.Integer.valueOf(String)
double 혹은 Double	java.lang.Double.valueOf(String)
float 혹은 Float	java.lang.Float.valueOf(String)
long 혹은 Long	java.lang.Long.valueOf(String)

JDBC LoginBean의 창조와 배비

이제는 회원싸이트에서 등록가입을 조종하는 JDBC LoginBean을 창조한다. 목록 5-7과 5-8의 실례는 JSP와 JavaBeans에 기초한 등록가입폼조종자의 기본원리를 제공한다. 목록 5-5의 LoginForm.jsp를 리용할 때 사용자는 자기의 이름과 통과암호를 입력하고 ProcessLogin.jsp를 호출하기 위하여 <SUBMIT>단추를 찰칵하였다.

이제 작성하는 ProcessLoginBean.jsp는 목록 5-5에서 보여준 JSP페지의 확장판이다. LoginForm.jsp와 ProcessLoginBean.jsp의 기본 차이점은 다음과 같다.

- ProcessLoginBean.jsp는 HTML부분이 없으며 순수 조종자로 동작한다.
- <jsp:forward />태그는 MVC구조의 뷰부분을 현시하는데 리용된다.

목록 5-7에서 보여준바와 같이 ProcessLoginBean.jsp는 업무론리를 다루는 JavaBean에 의거하고있다.

ProcessLoginBean.jsp페지가 JavaBean의 기동과 응답해석을 축소하기때문에 결과적인 MVC조종자는 단순하며 쉽게 리해할수 있다. 폼에서 입력된 내용은 LoginBean에 넘겨지며 사용자의 등록가입상태에 따라 사용자가 가야 할 세가지 페지들중 하나가 선택된다. 결과적인 JSP코드를 목록 5-9에 보여준다.

목록 5-9. ProcessLoginBean.jsp

```
<%@ page language="java"%>
<jsp:useBean id = "LoginBean" class = "JavaDB_Bible.ch05.sec02.
LoginBean"/>
<jsp:setProperty name="LoginBean" property="*"/>
<%
  String status = LoginBean.validate();
  String nextPage = "MemberWelcome.jsp";
  if(status.equals("New Member")) nextPage = "NewMemberForm.jsp";
  if(status.equals("Bad Password")) nextPage = "BadPasswordForm.jsp";
%>
<jsp:forward page="<%=nextPage%>"/>
```

LoginBean은 일정한 량의 HTML생성과 기타 부가적인 처리가 혼합된 써블레트와 달리 단순한 론리블로크이다. 속성에 대한 설정은 bean의 구체례가 만들어질 때 조종되며 JSP페지는 사용자이름과 통과암호를 설정한다.

목록 5-10. LoginBean

```
package JavaDB_Bible.ch05.sec02;

import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginBean extends java.lang.Object {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   private Connection con = null;
   protected String username;
```

```
    protected String password;

    public LoginBean() {
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String validate() {
        int id = -1;
        String memberPwd = null;

        try {
            Class.forName("com.inet.pool.PoolDriver");
            com.inet.tds.TdsDataSource tds =
                    new com.inet.tds.TdsDataSource();
            tds.setServerName("DOLPHIN");
            tds.setDatabaseName("MEMBERS");
            tds.setUser(dbUserName);
            tds.setPassword(dbPassword);

            DataSource ds = tds;
            Connection con = ds.getConnection(dbUserName, dbPassword);

            Statement stmt;
            ResultSet rs = null;

            String SQLQuery = "SELECT * FROM LOGIN WHERE Username = '" +
                              username + "';";
```

```
            stmt = con.createStatement();
            rs = stmt.executeQuery(SQLQuery);
            while (rs.next()) {
                memberPwd = rs.getString("Password");
                id = rs.getInt("MemberID");
            }
            con.close();
            stmt.close();
        } catch (ClassNotFoundException e1) {
            System.err.println(e1.getMessage());
        } catch (SQLException e2) {
            System.err.println(e2.getMessage());
        }

        if (id == -1) {
            return "New Member";
        } else if (!memberPwd.equals(password)) {
            return "Bad Password";
        } else {
            return "Member#" + id;
        }
    }
}
```

주의: JSP페지로부터 JavaBean의 속성들을 설정하는데 통용문자 "*"를 리용하기 위해서는 속성이름들이 HTML폼에서 리용된 변수이름들과 일치되여야 한다. 속성이름은 대소문자를 구별한다.

목록 5-10의 **LoginBean**은 사용자의 등록가입상태를 가리키는 문자렬을 돌려준다. 3가지 가능한 귀환값들은 다음과 같다.

- "New Member"
- "Bad Password"
- "Member#nnn"

JSP페지는 적당한 JSP페지로 사용자를 보내는것으로 매개 가능한 귀환값들을 처리한다. 등록된 회원인 경우 사용자는 단순히 기본싸이트의 Welcome페지로 가게 된다. 알지 못할 사용자이름으로 등록가입한 사람은 새로운 회원으로 인식하고 회원등록페지로 들어간다. 만일 사용자이름은 인식되였는데 통과암호가 틀리면 사용자에게 재시도, 새 회원으로 서명, 사용자의 전자우편주소로 정확한 통과암호 보내기의 추가선택항목이 제공된다.

제3절. PreparedStatement와 CallableStatement의 리용

지금까지 본 문제들과 실례들은 모두 SQL문을 실행하기 위하여 JDBC API를 어떻게 리용하는가 하는것이였다. SQL문이 자료기지관리체계에 넘겨졌을 때 실지로 어떤 일이 벌어지는가 하는데 대해서는 아직까지 론의하지 않았다.

기초적인 java.sql.Statement를 리용할 때에는 매번 기초적인 Statement객체가 실행되여 SQL지령이 관계형자료기지관리체계에 넘겨지고 관계형자료기지관리체계는 그 지령을 실행하기전에 분석하고 번역해야 한다.

SQL지령의 반복적인 분석과 번역으로 인한 간접부가시간을 없애기 위하여 JDBC는 미리 번역된 SQL문들을 리용하는 두가지 방법을 제공한다. 그것들은 PreparedStatement객체와 CallableStatement객체의 리용이다.

PreparedStatement와 CallableStatement의 리용은 웨브싸이트에 대한 폼처리와 같이 특정한 SQL지령이 자주 반복적으로 수행될 때 응용프로그람의 효률성을 크게 증가시킨다.

Statement객체의 3가지 특징들은 각이한 경우들을 상정한것이다. 첫번째 경우는 사용자가 문을 꼭 한번만 수행하고싶은 경우로서 이것이 표준적인 java.sql.Statement리용의 전형적인 경우이다. 만일 SQL문을 순환속에서 반복적으로 수행하고 그 다음에는 그것을 버려도 된다면 관계형자료기지관리체계에 의하여 분석 및 번역되여 림시로 완충기억되는 PreparedStatement를 리용하는것이 가장 좋은 방법이다. 끝으로 자주 집행하려는 문이나 문들의 묶음이 있는 경우에는 필요할 때마다 이름에 의해 호출될수 있도록 미리 번역되여 관계형자료기지관리체계에 항시적으로 저장되여있는 CallableStatement를 리용하는것이 리상적이다.

이 절에서는 SQL문의 실행성능을 개선하여 Java자료기지프로그람의 성능을 높일수 있는 두가지 중요한 방법들을 취급하게 된다.

5.3.1. PreparedStatement의 창조와 리용

기초적인 Statement객체와 PreparedStatement객체사이의 주되는 차이는 PreparedStatement가 리용되는 경우 SQL지령은 PreparedStatement가 창조될 때 자료기지관리체계에 넘겨지며 따라서 그것은 미리 번역되여 완충기억기에 보관된다는것이다. PreparedStatement를 실행할 때 그것은 일단 다시 분석되지만 재번역은 일어나지 않는다. 대신 이미 번역된것을 완충기억기에서 찾아 사용한다. 순환에서 SQL지령을 반복적으로 수행할것을 요구하는 프로그람인 경우 PreparedStatement를 리용하면 자료기지의 성능을 개선할수 있다.

PreparedStatement객체의 창조

PreparedStatement객체는 Statement객체와 마찬가지로 Connection을 리용하여 창조된다. 실례로 목록 5-10에서 개발한 LoginBean에서 Statement를 PreparedStatement로 교체해보자. 목록 5-11은 그 결과를 보여준다.

목록 5-11. PreparedStatement의 리용

```
Class.forName("com.inet.pool.PoolDriver");
com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
tds.setServerName( "DOLPHIN" );
tds.setDatabaseName( "MEMBERS" );
tds.setUser( dbUserName );
tds.setPassword( dbPassword );
DataSource ds = tds;

Connection con = ds.getConnection(dbUserName,dbPassword);
String SQLQuery = "SELECT * FROM LOGIN WHERE UserName = ?;";
PreparedStatement pstmt = con.prepareStatement(SQLQuery);
pstmt.setString(1, username);
ResultSet rs = pstmt.executeQuery(};
while(rs.next()){
   memberPwd = rs.getString("Password");
   id = rs.getInt("MemberID");
}
con.close();
```

이 실레에서 리용한 PreparedStatement와 2절에서 리용한 Statement객체의 주되는 차이는 SQL지령의 형식에 있다. 이 실레에서는 pstmt.setString()메쏘드를 리용하여 설정되는 기호 "?"가 변수 UserName에 대한 자리유지자로 사용되였다.

PreparedStatement를 실행하기전에 모든 자리유지자들에 해당한 값들을 할당해야 한다. 일단 PreparedStatement파라메터가 주어진 값으로 설정되면 다른 값으로 재설정하거나 clearParameters메쏘드가 호출되기전까지 그 값을 유지한다.

순환에서 PreparedStatement의 리용

PreparedStatement객체들을 리용하는데서 나타나는 실지 효과성은 그것들을 반복적으로 리용할 때(실레로 순환부분에서 SQL지령을 수행할 때) 나타난다. 만일 각이한 Java클라스의 구체례들로부터 같은 SQL지령을 자주 리용할 필요가 있다면 CallableStatement를 리용하는것이 더 좋은 선택안으로 된다.

순환에서 PreparedStatement를 리용하는 실레를 목록 5-12에서 보여주었다. 실레에서 간단한 for순환은 Orders배렬로부터 PreparedStatement의 파라메터들을 설정한다. 다음 자료가 3장의 실레에서 본것과 류사한 Ordered_Table표에 삽입된다.

목록 5-12. 순환에서 PreparedStatement의 리용

```
package JavaDB_Bible.ch05.sec03;

import java.sql.*;
import javax.sql.*;

public class PStatement {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   public static void main(String args[]) {
      int[][] Orders = { {1001, 327, 2}, {1001, 412, 1}, {1001, 906, 5},
                 {1002, 111, 7}, {1002, 112, 19} };

      try {
         Class.forName("com.inet.pool.PoolDriver");
         com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
         tds.setServerName("DOLPHIN");
         tds.setDatabaseName("CUSTOMERS");
         tds.setUser(dbUserName);
         tds.setPassword(dbPassword);

         DataSource ds = tds;
         Connection con = ds.getConnection(dbUserName, dbPassword);

         String SQLCmd = "INSERT INTO ORDERED_ITEMS(ORDER_NUMBER, " +
                   "ITEM_NUMBER, QTY) VALUES(?,?,?)";
         PreparedStatement pstmt = con.prepareStatement(SQLCmd);

         for (int i = 0; i < 5; i++) {
            pstmt.setInt(1, Orders[i][0]);
            pstmt.setInt(2, Orders[i][1]);
            pstmt.setInt(3, Orders[i][2]);
            pstmt.executeUpdate();
         }

         con.close();
      } catch (ClassNotFoundException e1) {
         System.err.println(e1.getMessage());
      } catch (SQLException e2) {
         System.err.println(e2.getMessage());
      }
   }
}
```

순환을 수행한 다음 Ordered_Items표는 표 5-8과 같이 된다. ID렬은 자동증가자료형을 리용하였으며 우의 실례에서는 특별히 반영하지 않았다.

표 5-8. Ordered_Items표

ID	Order_Number	Item_Number	Qty
1	1001	327	2
2	1001	412	1
3	1001	906	5
4	1002	111	7
5	1002	112	19

PreparedStatement에 의해 귀환되는 값

PreparedStatement가 돌려줄수 있는 값들의 종류는 기본 Statement와 같다. 목록 5-11에서 리용한 executeQuery()메쏘드는 질문의 결과를 포함하는 ResultSet객체를 돌려주었지만 목록 5-12의 executeUpdate()메쏘드가 돌려주는 값은 표에서 갱신된 행들의 수를 가리키는 옹근수이다. 실례로 목록 5-12의 pstmt.executeUpdate()를 아래에서 보여주는 바와 같이 조종탁에 현시할수도 있다.

```
System.out.println(pstmt.executeUpdate());
```

만일 executeUpdate()메쏘드가 CREATE TABLE과 같은 DDL문을 수행하는데 리용된다면 령을 돌려준다.

5.3.2. CallableStatement의 창조와 리용

CallableStatement객체는 Java프로그람으로부터 자료기지저장수속을 호출할수 있게 한다. CallableStatement객체는 PreparedStatement객체와 아주 류사하며 실지 PreparedStatement의 확장이다.

그러나 PreparedStatement객체가 SQL지령을 실제로 포함하고있다면 CallableStatement객체는 자료기지에 저장되여있는 수속에 대한 호출만을 포함하고있으며 저장수속 그 자체를 포함하지는 않는다. 수많은 사용자들이 같은 SQL문을 반복적으로 수행하는 웨브싸이트와 같은 응용프로그람에서 CallableStatement를 리용하면 자료기지의 성능을 크게 개선할수 있다.

CallableStatement는 PreparedStatement의 확장이기때문에 Prepared

Statement객체가 가질수 있는 입력파라메터들을 다 가질수 있을뿐아니라 출력파라메터 혹은 입력파라메터와 출력파라메터를 둘다 가질수 있다.

입력파라메터들은 SQL의 CREATE PROCEDURE문에서 다음과 같은 문법을 리용하여 정의된다.

```
@파라메터이름 형[(크기)]
```

"@"기호는 뒤에 오는 파라메터이름을 SQL엔진을 위한 파라메터이름으로 식별하게 한다. 형마당과 크기마당은 표를 창조하는데 쓰이는 표준 SQL자료형마당들과 대응된다.

저장수속의 창조

회원웨브싸이트실례로 되돌아가서 본다면 사용자가 진행하는 첫 단계사업은 우선 그 싸이트에 가입하는것이다. 사용자이름과 통과암호가 인식되지 않는 경우 사용자에게는 새로운 회원으로 등록할수 있는 기회가 차례진다.

새 회원의 가입과 관련하여 갱신되여야 할 첫 표는 Contact_Info표이다. 이 표는 새 회원이 새회원등록폼에 대한 입력을 끝내면 갱신된다. 회원등록폼에 대한 입력을 끝낸 다음 회원지망자들은 계속해서 추가적인 폼들에로 이동하게 된다. 이것들은 Contact_Info표와 Member_Profile표들에서 회원의 기입사항들을 완성하는데 리용된다.

Contact_Info표에는 그 회원의 실지이름과 집주소, 전자우편주소와 같은 자료들이 저장된다. Contact_Info표를 표 5-9에 보여주었다.

표 5-9. Contac t_Info표

ID	Family_ Name	Personal_ Name	State	City	Street	Phone	Email
1	김	성철	평양시	평양	창광거리	359-6532	tiger@home.com
2	김	은희	평양시	평양	천리마거리	546-2259	bing@cn.com

Contact_Info표의 갱신은 보통 반복되는 처리이기때문에 저장수속을 리용하는것이 좋다. 저장수속을 실행하는데 CallableStatement객체가 리용된다. 목록 5-13에서는 Contact_Info표에 자료를 이식하기 위한 저장수속의 작성방법을 보여준다.

목록 5-13. 저장수속의 창조

```
package JavaDB_Bible.ch05.sec03;

import java.sql.*;
import javax.sql.*;

public class CreateCallableStmt {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   public static void main(String args[]) {
      String createProc = "CREATE PROCEDURE INSERT_CONTACT_INFO " +
                        "@ID INT, @FName VARCHAR(20), " +
                        "@PName VARCHAR(30), @State VARCHAR(30), " +
                        "@City VARCHAR(30), @Street VARCHAR(30), " +
                        "@Phone VARCHAR(30), @Email VARCHAR(30) " +
                        "AS INSERT INTO CONTACT_INFO " +
                        "(MemberID, Family_Name, Personal_Name, " +
                        "State, City, "Street, Phone, Email) " +
                        "VALUES " +
                        "(@ID, @FName, @PName, @State, @City, " +
                        " @Street, @Phone, @Email);";

      try {
         Class.forName("com.inet.pool.PoolDriver");
         com.inet.tds.TdsDataSource tds =
               new com.inet.tds.TdsDataSource();
         tds.setServerName("DOLPHIN");
         tds.setDatabaseName("MEMBERS");
         tds.setUser(dbUserName);
         tds.setPassword(dbPassword);

         DataSource ds = tds;
         Connection con = ds.getConnection(dbUserName, dbPassword);
         Statement stmt = con.createStatement();
         stmt.executeUpdate(createProc);
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }
}
```

저장수속의 호출

저장수속들은 대괄호로 호출구문을 둘러싸는 단순한 탈출문장구성법을 리용하여 호출된다. 다음의 코드는 목록 5-13에서 만든 저장수속이 어떻게 호출되는가를 보여준다.

```
String[] newMember = {"리", "철수", "평안남도", "평성",
                     "강성거리", "123-406-7890", "fat@pizza.com"};

CallableStatement cs = con.prepareCall("{call INSERT _ CONTACT_INFO
(?,?,?,?,?,?,?,?)}");

cs.setInt(1, 7);
for(int i=0; i<newMember.length; i++){
    cs.setString(i+2,newMember[i]);
}
System.out.println(cs.executeUpdate() + " row updated");
```

CallableStatement객체는 저장수속이 호출될 때 Connection의 메쏘드 prepareCall()에 의해 창조된다. prepareCall()메쏘드의 인수는 저장수속을 호출하는 탈출문자렬이다. 구동프로그람은 대괄호를 만나는 경우 그 안에 들어있는 지령을 저장수속을 호출하기 위하여 자료기지에 의해 리용되는 순수한 SQL문으로 번역한다.

저장수속의 인수로 포함된 물음기호(?)는 저장수속을 작성할 때 <@이름> 약속을 리용하여 정의된 입력파라메터들에 대한 자리유지자이다. IN파라메터값들은 PreparedStatement로부터 계승된 설정자메쏘드들에 의해 설정되며 그 다음에 CallableStatement가 수행된다.

이번에는 ResultSet를 돌려주는 저장수속의 호출실례를 보자. 주어진 사용자이름에 대한 등록가입자료를 얻는 간단한 저장수속 GET_LOGIN_FOR_USER는 다음과 같이 정의할수 있다.

```
CREATE PROCEDURE GET_LOGIN_FOR_USER
@USERNAME VARCHAR(20)
AS SELECT *
   FROM LOGIN
   WHERE USERNAME = @USERNAME;
```

목록 5-14는 저장수속 GET_LOGIN_FOR_USER를 호출하는 방법을 보여준다.

목록 5-14. ResultSet를 돌려주는 저장수속의 호출

```
package JavaDB_Bible.ch05.sec03;

import java.sql.*;
import javax.sql.*;

public class CallableGetLogin {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   public static void main(String args[]) {
      try {
         Class.forName("com.inet.pool.PoolDriver");
         com.inet.tds.TdsDataSource tds =
                 new com.inet.tds.TdsDataSource();
         tds.setServerName("DOLPHIN");
         tds.setDatabaseName("MEMBERS");
         tds.setUser(dbUserName);
         tds.setPassword(dbPassword);

         DataSource ds = tds;
         Connection con = ds.getConnection(dbUserName, dbPassword);

         CallableStatement cs = con.prepareCall(
               "{call GET_LOGIN_FOR_USER(?)}");
         cs.setString(1, "RyoMiSong");

         ResultSet rs = cs.executeQuery();
         ResultSetMetaData md = rs.getMetaData();

         while (rs.next()) {
            for (int i = 1; i <= md.getColumnCount(); i++) {
               System.out.print(md.getColumnLabel(i) + " = ");
               if (md.getColumnType(i) == java.sql.Types.INTEGER)
                  System.out.println(rs.getInt(i));
               else
                  System.out.println(rs.getString(i));
            }
         }
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

일어나는 사건들은 앞에서 본 실례에서와 같다.

1. prepareCall()을 리용하여 CallableStatement가 창조된다.
2. CallableStatement의 파라메터들이 설정된다.
3. CallableStatement구체례가 수행된 다음 ResultSet가 귀환된다.

이 실례에서는 자료검색에 정당한 취득자메쏘드가 리용되였는가를 확인하기 위하여 귀환값들에 대한 간단한 형검사를 하고있다.

JSP Bean에서 저장수속의 리용

저장수속이 창조되면 그것은 JSP페지에서 구체례가 작성된 JavaBean으로부터 호출된다. JSP페지자체는 회원등록처리의 부분으로 현시되는 HTML폼의 action메쏘드에서 호출된다. 폼 자체는 그림 5-9에, 이 폼을 만드는 HTML은 목록 5-15에 보여주었다.

Information contained in the shaded portion of this page be kept confidential.

Family Name

Personal Name

Choose a User Name

Choose a password

EMail Address

Phone

Street Address

Information entered below this line will be used by search engine.

City

State/Province

Please choose

Click here to proceed

그림 5-9. Contact_Info표에 대한 자료를 얻어내는 HTML폼

코드에서 매개 중요한 마당들에는 어떤 자료든지 입력되여야 한다는것을 담보하기 위하여 간단한 확인스크립트가 포함되여있다. 이 폼은 NewMemberForm.jsp로 보관되고 목록 5-9에 보여준 폼조종기인 ProcessLogin.jsp에서 호출된다.

목록 5-15. 회원등록폼 NewMemberForm.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>
    Member Registration
  </TITLE>
  <SCRIPT language="JavaScript 1.1" type="">
  function validate(form){
    if(form.elements["familyName"].value == "" ||
       form.elements["personalName"].value == "" ||
       form.elements["email"].value == "" ||
       form.elements["city"].value == "" ||
       form.elements["state"].value == "?"){
      alert("Please enter your name, email, city and state.");
      return false;
    }
    return true;
  }
  </SCRIPT>
  <META content="text/html; charset=UTF8" http-equiv=Content-Type>
</HEAD>

<BODY bgColor=#ffffff>
<BASEFONT face=Arial size=3>
<FORM action="ProcessNAForm.jsp" method=POST target=_self
  onSubmit="return validate(this);">
  <TABLE cellPadding=0 border=1>
    <TR>
    <TD>
      </P>
      <SMALL>
      <CENTER>
        <FONT color=#ff0000>
        Information contained in the shaded portion
        of this page will be kept confidential.
```

```
        </FONT>
      </CENTER>
      </SMALL>
    </TD>
    </TR>
    <TR>
    <TD>
      <TABLE cellPadding=4 cellSpacing=0 bgcolor="#AAAAAA" width="100%">
      <TBODY>
        <TR>
          <TD vAlign=bottom>
          Family Name<BR>
          <INPUT maxLength=30 name=familyName size=30>
          </TD>
          <TD vAlign=bottom>
          Personal Name<BR>
          <INPUT maxLength=30 name=personalName size=30>
          </TD>
        </TR>
        <TR>
          <TD vAlign=bottom>
          Choose a User Name<BR>
          <INPUT maxLength=30 name=username size=30>
          </TD>
          <TD height=43 colspan=2 vAlign=bottom>
          Choose a password<BR>
          <INPUT name=password size=20>
          </TD>
        </TR>
        <TR>
          <TD vAlign=bottom>
          EMail Address<BR>
          <INPUT maxLength=30 name=email size=30>
          </TD>
          <TD vAlign=bottom>
          Phone<BR>
          <INPUT maxLength=30 name=phone size=30>
          </TD>
        </TR>
        <TR>
          <TD height=43 vAlign=bottom>
          Street Address<BR>
          <INPUT name=Street size=30>
```

```
        </TD>
      </TR>
    </TBODY>
    </TABLE>
    </TD>
  </TR>
  <TR>
  <TD>
    </P>
    <SMALL>
    <CENTER>
    <FONT color=#ff0000>
    Information entered below this line will be used by our search engine.
    </FONT>
    </CENTER>
    </SMALL>
  </TD>
  </TR>
  <TR>
  <TD>
    <TABLE cellPadding=4 cellSpacing=0>
    <TBODY>
      <TR>
        <TD height=43 vAlign=bottom width=256>
        City<BR>
        <INPUT name=City size=20>
        </TD>
        <TD height=49 vAlign=bottom width=257>
        State/Province<BR>
        <select name=State size=1>
          <option selected value = "?">Please choose</option>
          <option value=PY>PyongYangSi</option>
          <option value=PN>PyongAnNamDo</option>
          <option value=PB>PyongAnBukDo</option>
          <option value=JG>JaGangDo</option>
          <option value=RG>RyangGangDo</option>
          <option value=GW>GangWonDo</option>
          <option value=GN>HamGyongNamDo</option>
          <option value=GB>HamGyongBukDo</option>
          <option value=HN>HwangHaeNamDo</option>
          <option value=HB>HwangHaeBukDo</option>
          <option value=GS>GaeSongSi</option>
          <option value=RS>RaSonSi</option>
        </select>
```

```
          </TD>
        </TR>
      </TBODY>
      </TABLE>
    </TD>
    </TR>
    <TR>
    <TD align=center colspan=3>
      <BR/>
      <INPUT name=SubmitButton type=SUBMIT value="Click here to proceed">
      <BR/>
      <P/>
    </TD>
    </TR>
  </TABLE>
</FORM>
</BODY>
</HTML>
```

JavaScript가 국부적인 정당성을 검사한 다음 폼자료는 자료기지에 삽입되기 위하여 ProcessNABean을 리용하는 JSP페지 ProcessNAForm.jsp에 넘겨진다. ProcessNAForm.jsp는 JSP폼조종자의 간단한 실례이다. ProcessNAForm.jsp는 ProcessNABean을 적재하고 setProperty메쏘드에서 "*"를 리용하여 그의 속성들을 설정한다. setProperty메쏘드는 폼자료로부터 JavaBean의 모든 속성들을 설정하기 위하여 자기판찰기능에 의거한다. insertData()메쏘드가 호출되면 ProcessNABean은 문자렬 nextPage를 설정하는데 리용되는 론리형값을 돌려준다. 끝으로 <jsp:forward />태그는 사용자를 해당한 페지로 인도하는데 리용된다. 목록 5-16은 ProcessNAForm.jsp를 보여준다.

목록 5-16. ProcessNAForm.jsp

```
<%@ page language="java"%>
<jsp:useBean id="ProcessNABean"
   class="JavaDB_Bible.ch05.sec03.ProcessNABean" scope="session"/>
<jsp:setProperty name="ProcessNABean" property="*"/>
<%
   String nextPage = "MemberWelcome.jsp";
   if(ProcessNABean.insertData()){
      nextPage = "MemberProfile.jsp";
   }else{
      nextPage = "NewMemberForm.jsp";
```

```
    }
%>
<jsp:forward page="<%=nextPage%>"/>
```

ProcessNABean의 작용

ProcessNABean의 첫 부분에는 bean의 파라메터들에 접근하는데 필요한 취득자메쏘드와 설정자메쏘드들을 모아놓았다. 이것들은 JSP엔진이 작업하는데 필요한 bean의 자기관찰기능을 위한것이다.

자료기지에 자료를 써넣는 실지 작업은 ProcessNABean의 insertData()메쏘드에서 진행된다. ProcessNABean은 CallableStatement객체 cs를 널리 리용하고있다. 먼저 사용자가 입력한 UserName이 Login표에 있는가를 보기 위하여 GET_LOGIN_FOR_USER수속을 호출한다. 만일 사용자가 입력한 UserName을 다른 사용자가 리용하고있는 경우 론리형기발 username_selection_ok를 false로 설정하여 사용자에게 다른 UserName을 선택할것을 요구하는 통보문을 내보낸다.

일단 사용자가 정당하면서도 유일한 UserName을 선택하면 새로운 UserName과 Password를 가지고 Login표를 갱신하기 위하여 저장수속 SET_LOGIN_FOR_USER를 호출한다. 그 저장수속은 다음과 같다.

```
CREATE PROCEDURE SET_LOGIN_FOR_USER
    @USERNAME VARCHAH(20),
    @PASSWORD VARCHAR(20)
    AS
    INSERT INTO LOGIN (USERNAME, PASSWORD)
        VALUES(@USERNAME, @PASSWORD);
```

그 다음에는 이 사용자에게 할당된 자동생성 MemberID를 얻기 위하여 저장수속 GET_LOGIN_FOR_USER가 다시 호출된다. ID를 얻는 방법에는 아래에 보여준것처럼 Statement객체에 대하여 JDBC 3.0에서 정의된 getGeneratedKeys()메쏘드를 리용하는 보다 고급한 방법도 있다.

```
if(cs.executeUpdate()!=1) ok = false;
ResultSet rs = cs.getGeneratedKeys();
```

마지막으로 저장수속 INSERT_CONTACT_INFO가 ProcessNABean의 속성들에 저장되는 회원자료들을 삽입하기 위하여 호출된다. ProcessNABean의 코드를 목록 5-17에 보여준다.

목록 5-17. JavaBean에서 저장수속호출

```
package JavaDB_Bible.ch05.sec03;

import java.sql.*;
import javax.sql.*;

public class ProcessNABean extends java.lang.Object {
  private static String dbUserName = "sa";
  private static String dbPassword = "dba";

  protected String familyName;
  protected String personalName;
  protected String street;
  protected String city;
  protected String state;
  protected String phone;
  protected String email;
  protected String username;
  protected String password;

  public ProcessNABean() {
  }

  public void setUsername(String username) {
    this.username = username;
  }

  public void setPassword(String password) {
    this.password = password;
  }

  public void setFamilyName(String familyName) {
    this.familyName = familyName;
  }

  public void setPersonalName(String personalName) {
    this.personalName = personalName;
  }

  public void setStreet(String street) {
    this.street = street;
```

```
}

public void setCity(String city) {
   this.city = city;
}

public void setState(String state) {
   this.state = state;
}

public void setPhone(String phone) {
   this.phone = phone;
}

public void setEmail(String email) {
   this.email = email;
}

public String getUsername() {
   return username;
}

public String getPassword() {
   return password;
}

public String getFamilyName() {
   return familyName;
}

public String getPersonalName() {
   return personalName;
}

public String getStreet() {
   return street;
}

public String getCity() {
   return city;
}
```

```
public String getState() {
   return state;
}

public String getPhone() {
   return phone;
}

public String getEmail() {
   return email;
}

public boolean insertData() {
   boolean username_selection_ok = false;

   try {
      Class.forName("com.inet.pool.PoolDriver");
      com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
      tds.setServerName("DOLPHIN");
      tds.setDatabaseName("MEMBERS");
      tds.setUser(dbUserName);
      tds.setPassword(dbPassword);

      DataSource ds = tds;
      Connection con = ds.getConnection(dbUserName, dbPassword);

      CallableStatement cs = con.prepareCall(
            "{call GET_LOGIN_FOR_USER(?)}");
      cs.setString(1, username);
      ResultSet rs = cs.executeQuery();
      ResultSetMetaData md = rs.getMetaData();

      int id = -1;
      while (rs.next()) {
         id = rs.getInt("MemberID");
         if (id >= 0) {
            System.out.println(id + ": " + username + "; " + password);
            username_selection_ok = true;
         } else {
            cs = con.prepareCall("{call SET_LOGIN_FOR_USER(?,?)}");
            cs.setString(1, username);
            cs.setString(2, password);
```

```
            if (cs.executeUpdate() != 1) username_selection_ok = true;

            cs = con.prepareCall("{call GET_LOGIN_FOR_USER(?)}");
            cs.setString(1, username);
            rs = cs.executeQuery();
            while (rs.next()) {
               id = rs.getInt("MemberID");
            }

            cs = con.prepareCall(
                 "{call INSERT_CONTACT_INFO(?,?,?,?,?,?,?,?)}");
            cs.setInt(1, id);
            cs.setString(2, familyName);
            cs.setString(3, personalName);
            cs.setString(4, state);
            cs.setString(5, city);
            cs.setString(6, street);
            cs.setString(7, phone);
            cs.setString(8, email);
            if (cs.executeUpdate() != 1) username_selection_ok = true;
         }
      }
   } catch (ClassNotFoundException e1) {
      System.err.println(e1.getMessage());
   } catch (SQLException e2) {
      System.err.println(e2.getMessage());
   }
   return username_selection_ok;
  }
}
```

오유조종

이미 설명하였지만 ProcessNABean은 론리형기발 username_selection_ok를 false로 설정하여 사용자가 다른 UserName을 선택해야 한다는것을 ProcessNAForm.jsp페지에 통지한다. 그러면 ProcessNAForm.jsp는 문제가 생겼다는것을 알고 사용자가 새로운 사용자이름과 통과암호를 선택할수 있도록 다시 폼에 돌려보낸다.

그런데 이때 폼이 다시 현시되면 이미 입력된 내용이 다 없어진다. 이렇게 되면 사용자는 그 모든 자료들을 다시 입력해야 하므로 불편을 느낄수 있다. 이것을 피하는 방도는 사용자가 이미 완성한 마당들은 그대로 놔두고 사용자가 이제 무엇을 해야 하는가를 말해주는 통보문을 현시하는것이다.

JSP응용에서 JavaBean들을 리용하는 기본목적의 하나는 자료저장이다. 모든 폼자료가 이미 ProcessNABean에 삽입되였으므로 JSP페지에 다음의 행만 추가하면 사용자를 위한 폼이 완성된다.

```
<jsp:useBean id="ProcessNABean" .../>
```

또한 속성들을 설정하기 위한 몇개의 행들도 포함시킨다.

```
First Name<BR><INPUT maxLength=30 name=firstName
value='<jsp:getProperty  name="ProcessNABean"  property="firstName"/>'
size=26>
```

변경된 폼의 부분적인 목록을 목록 5-18에 보여주었다.

목록 5-18. 오유페지로 리용하기 위하여 변경한 NewMemberForm.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE>
      Member Registration
    </TITLE>
    <script language="JavaScript l.1" type="">
      function validate(form){
        if(form.elements["familyName"].value == "" ||
        form.elements["personalName"].value == "" ||
        form.elements["email"].value == "" ||
        form.elements["city"].value == "" ||
        form.elements["state"].value == "?"){
          alert("Please enter your name, email, city and state.");
          return false;
        }
        return true;
      }
    </script>
    <META content="text/html; charset=big5" http-equiv=Content-Type>
  </HEAD>

  <BODY bgcolor=#ffffff>
  <BASEFONT face=Arial size=3>
    <%@page session="true" %>
    <jsp:useBean id="ProcessNABean"
```

```
       class="JavaDB_Bible.ch05.sec03.ProcessNABean"
       scope="session"/>
<FORM action="ProcessNAForm.jsp" method="POST"
      target="_self" onSubmit="return validate(this);">
<TABLE cellPadding=0 border=0>
<TR>
  <TD>
  <TABLE cellPadding=0 border=1>
  <TR>
  <TD>
    </p>
    <small>
    <CENTER>
      <FONT color=#ff0000>
        Information contained in the shaded portion of
        this page be kept confidential.
      </FONT>
     </CENTER>
     </small>
  </TD>
</TR>
<TR>
  <TD>
    <TABLE cellPadding=4 cellSpacing=0 border=0
      bgcolor="#AAAAAA" width="100%">
    <TBODY>
      <TR>
      <TD vAlign=bottom>
        Family Name<BR>
        <INPUT maxLength=30 name=familyName value=
        '<jsp:getProperty name="ProcessNABean" property="familyName"/>'
        size=26>
      </TD>
      <TD vAlign=bottom>
        Personal Name<BR>
        <INPUT maxLength=30 name=personalName value=
        '<jsp:getProperty name="ProcessNABean" property="personalName"/>'
        size=30>
      </TD>
      </TR>
      <TR>
      <TD vAlign=bottom>
```

```
        Choose a User Name<BR>
        <INPUT maxlength=30 name=username size=30>
      </TD>
      <TD height=43 colspan=2 vAlign=bottom>
        Choose a password<BR>
        <INPUT maxlength=20 name=password size=20>
      </TD>
      </TR>
      <TR>
        <TD vAlign=bottom>
          EMail Address<BR>
          <INPUT maxLength=30 name=email value=
          '<jsp:getProperty name="ProcessNABean" property="email"/>'
          size=30>
        </TD>
        <TD vAlign=bottom>
          Phone<BR>
          <INPUT maxLength=30 name=phone value=
          '<jsp:getProperty name="ProcessNABean" property="phone"/>'
          size=30>
        </TD>
      </TR>
      <TR>
        <TD height=43 vAlign=bottom>
          Street Address<BR>
          <INPUT name=street value=
          '<jsp:getProperty name="ProcessNABean" property="street"/>'
          size=30>
        </TD>
      </TR>
    </TBODY>
    </TABLE>
  </TD>
</TR>
<TR>
  <TD>
    </p>
    <small>
    <CENTER>
    <FONT color=#ff0000>
    Information entered below this line will be used by search engine.
    </FONT>
```

```
    </CENTER>
    </small>
  </TD>
</TR>
```

주의: 목록 5-18은 JSP의 기본 작업부분만을 보여주고있다. 이것을 목록 5-15와 결합하여 완성해야 한다.

그림 5-10은 사용자와 호상작용하는 폼을 보여주고있다. 이러한 호상작용방식은 사용자의 편리성을 중시하는 관점에서 아주 중요하다.

Information contained in the shaded portion of this page be kept confidential.

Family Name: 리
Personal Name: 철수
Choose a User Name:
Choose a password:
EMail Address: fat@pizza.com
Phone: 123-406-7890
Street Address: 강성거리

Information entered below this line will be used by search engine.

City: 평성
State/Province: 평안남도

Click here to proceed

그림 5-10. 먼저 입력한 자료들을 그대로 현시하는 회원등록폼

입력 및 출력파라메터들을 가진 저장수속의 리용

저장수속에 입력파라메터들을 줄뿐아니라 저장수속에서 출력파라메터들을 얻을수도 있다. 출력파라메터를 리용하려면 아래와 같이 execute메쏘드가 호출되기전에 CallableStatement.registerOutParameter()메쏘드를 써서 그것을 OUT파라메터로 등록해야 한다.

```
cstmt.registerOutParameter(1, java.sql.Types.VARCHAR);
```

OUT파라메터값들은 그 값들의 자료형에 맞는 취득자메쏘드들이 수행된 후에 검색될수 있다. 목록 5-19는 자료기지에서 사용자의 이름과 통과암호를 찾게 되면 "PASS"문자렬을, 그렇지 않으면 "FAIL"문자렬을 귀환하는 간단한 저장수속의 실례를 보여주고있다.

목록 5-19. 저장수속과 함께 출력파라메터의 리용

```
CREATE PROCEDURE CHECK_USER_NAME
   @UserName varchar(30),
   @Password varchar(20),
   @PassFail varchar(20) OUTPUT
   As
   IF EXISTS(Select * From Login Where UserName = @UserName
         And Password = @Password)
      SELECT @PassFail = 'PASS'
   else
      SELECT @PassFail = 'FAIL';
```

주의: 저장수속은 여러개의 SQL문을 포함할수 있으며 그것들이 여러개의 결과들을 내보내는 경우에는 execute메쏘드를 리용하여야 한다. CallableStatement객체가 여러개의 ResultSet객체들을 돌려주는 경우 모든 결과들은 OUT파라메터들이 검색되기전에 먼저 getMoreResults메쏘드를 써서 검색되여야 한다.

목록 5-20은 목록 5-19의 저장수속을 리용하는 실례를 보여준다. registerOutParameter()메쏘드에 대한 호출이 출력파라메터를 검색하기 위한 CallableStatement의 getString메쏘드의 호출보다 앞서 진행되였다는데 대하여 주의하기 바란다.

목록 5-20. 저장수속로부터 출력파라메터의 얻기

```
package JavaDB_Bible.ch05.sec03;

import java.sql.*;
import javax.sql.*;

public class CheckPassword {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   public static void main(String args[]) {
      int id = -1;
      String password = null;
      String username = "";

      if (args.length > 1) {
         username = args[0];
         password = args[1];
         try {
            Class.forName("com.inet.pool.PoolDriver");
            com.inet.tds.TdsDataSource tds =
                   new com.inet.tds.TdsDataSource();
            tds.setServerName("DOLPHIN");
            tds.setDatabaseName("MEMBERS");
            tds.setUser(dbUserName);
            tds.setPassword(dbPassword);

            DataSource ds = tds;
            Connection con = ds.getConnection(dbUserName,
                                     dbPassword);

            CallableStatement cs = con.prepareCall(
                 "{call CHECK_USER_NAME(?,?,?)}");
            cs.setString(1, username);
            cs.setString(2, password);
            cs.registerOutParameter(3, java.sql.Types.VARCHAR);
            cs.executeUpdate();
            System.out.println(cs.getString(3));
         } catch (ClassNotFoundException e) {
            e.printStackTrace();
         } catch (SQLException e) {
            e.printStackTrace();
         }
      } else {
         System.out.println(
              "Enter yout username & password into this line");
```

```
        }
    }
}
```

제4절. Blobs와 Clobs를 리용한 대형객체관리

원래 관계형자료기지관리체계는 바이트형, 옹근수형, 실수형, 문자형과 같은 간단한 자료형들에 대한 취급을 념두에 두고 설계되였다. 그러나 콤퓨터하드웨어와 쏘프트웨어의 발전은 많은 큰 자료객체 즉 화상과 지어는 비데오클립까지도 경제적이며 효률적으로 저장할수 있게 하였다.

지난시기에는 대형자료객체들을 전통적인 파일체계에 저장하였으며 그 수가 대단히 많을 때에는 효률에서 큰 손실을 보았다. 관계형자료기지관리체계를 설계하는 사람들은 자료기지자체내에 이러한 대형객체들의 관리와 저장에 대한 지원을 제공함으로써 이 문제를 해결하였다.

이 절에서는 여러가지 방법으로 대형객체들을 저장하고 검색하기 위한 관계형자료기지의 리용에 대하여 론의한다.

5.4.1. 대형객체

대형객체(large object : LOB)에 대한 지원은 현대 객체관계형자료기지의 중요한 기능의 하나이다. SQL3규약은 대형객체들을 관리하기 위한 새로운 자료형들을 정의하고있다. 이 자료형들은 JDBC확장 API에 의해 제공된다. JDBC 2.0확장에 의하여 지원되는 새로운 SQL3 대형객체자료형들은 다음과 같다.

- ARRAY - 배렬을 하나의 렬값으로 저장할수 있다.
- BLOB(binary large object) - 대단히 많은 량의 자료를 그대로 바이트들로 저장할수 있다.
- CLOB(character large object) - 대단히 많은 문자자료를 저장할수 있다.
- 구조체형(Structured types)
- 구조체형에 대한 참조

경고: 서로 다른 관계형자료기지관리체계들은 대형객체들을 저장하는데 각이한 내부자료형들을 리용한다. 따라서 대형객체저장에 필요한 자료형들을 찾으려면 필요한 참고서들을 보아야 한다.

JDBC 2.0은 SQL3의 자료형들에 대응되는 대면부들의 모임을 정의한다. 표 5-10에서는 형대응과 각이한 대형객체들에 대한 검색, 저장 및 갱신메쏘드들을 보여준다.

표 5-10. SQL3 대형객체 자료형들

SQL3자료형	Java대면부	get	set	update
BLOB	java.sql.Blob	getBlob	setBlob	updateBlob
CLOB	java.sql.Clob	getClob	setClob	updateClob
ARRAY	java.sql.Array	getArray	setArray	updateArray
SQL 구조체형	java.sql.Struct	getObject	setObject	updateObject
구조체형에 대한 참조	java.sql.Ref	getObject	setObject	updateObject

대형객체의 지원은 가격, 날자, 수량들과 같은 전통적인 자료형들은 물론 화상과 같은 비전통적인 자료형들을 다루기 위한 요구를 만족시키기 위한데 있다. 전통적인 자료형들은 상대적으로 단순하며 일반적으로 옹근수값들을 저장하는 몇바이트의 자료로부터 이름이나 주소를 저장하는 수십바이트의 자료에 이르기까지 어디서나 다 요구된다. 지금까지 관계형자료기지관리체계는 이러한 적은 수의 자료마당형들을 포함하는 행들을 다루는데 최적화되였다.

오늘날의 많은 프로그람들에서는 수십KB의 화상으로부터 수백MB의 비데오화상에 이르기까지 훨씬 더 큰 자료객체들에 대한 관리를 요구하고있다.

대형객체들을 조종하기 위한 제일 초기의 방법은 그것들을 밑준위 OS파일로 저장하고 파일경로만을 자료기지에 보관하여 프로그람코드에서 파일들을 관리하게 하는것이였다. 오늘날 많은 기업용 관계형자료기지관리체계들에서는 대형객체들을 특수한 자료형으로 직접 지원해준다. 물론 그것들을 질문에서 리용하는데는 일정한 제한이 있다.

대형객체는 정의그대로 매우 크기때문에 SQL위치자를 리용하여 다룬다. 개념적으로 위치자는 객체자체가 아니라 그 객체의 위치를 보관하는 C나 C++에서의 지적자와 류사하다. 관계형자료기지관리체계는 대형객체들을 관리하는데 위치자를 리용한다. 왜냐하면 대형객체들을 직접 조종하는 경우 관계형자료기지관리체계가 자료객체들을 디스크분구와 같은 물리적저장장치들에 넘기는데서 실시하는 최적화가 파괴되기때문이다.

ARRAY, BLOB, CLOB의 중요한 특징이 바로 봉사기에서 의뢰기로 자료를 전부 복사하지 않고도 관리할수 있는 위치자를 리용하여 접근될수 있다는것이다. 사실상 자료기지에서 대형객체들에 대한 질문을 작성하면 실지 객체가 아니라 그의 위치자가 ResultSet에

귀환된다. 이것은 매개 마당에 대하여 대량의 자료를 직접 체계에서 움직이는것보다 훨씬 더 효과적이다.

JDBC개발자로서는 위치자를 다루는것이 아니지만 그 개념을 리해할 필요가 있으며 그로부터 다양한 대형객체조작메쏘드들의 동작방식을 알수 있다. 일단 위치자를 가지면 명확히 대형객체자료를 요구해야 한다. 이 처리를 자료의 **구체화**(materializing)라고 한다. 실례로 BLOB로 저장된 화상을 검색하려면 Blob.getBytes()메쏘드를 써서 바이트배렬로 구체화하든지 혹은 Blob.getBinaryStream()메쏘드를 써서 InputStream으로 구체화할수 있다.

표 5-10에서 알수 있는바와 같이 검색, 저장, 갱신메쏘드는 대형객체에 대해서도 다른 모든 자료형들과 꼭같이 지원된다.

Blob를 리용한 2진자료의 저장

Blob는 대량의 2진자료들을 저장하고 관리하는 한가지 수단을 제공한다. 대형2진자료의 대표적인 실례는 음향자료, 비데오자료 및 화상파일들이다.

Blob는 특히 웨브프로그람들에서 화상들을 저장하는데 쓸모가 있다. Blob에 대한 JDBC의 지원은 다음의 접근메쏘드들이 정의된 Blob대면부에 의해 제공된다.

- public InputStream getBinaryStream()
- public byte[] getBytes(long position, int length)

Blob대면부는 또한 편의메쏘드들인 length()와 position()메쏘드를 정의하고있다. length()메쏘드는 Blob의 바이트수를 돌려주며 position()메쏘드는 포함된 바이트배렬이나 Blob에 대한 편위를 돌려준다. ResultSet의 getBlob()메쏘드는 ResultSet에서 Blob의 위치자를 얻는데 리용되며 PreparedStatement대면부의 setBlob()메쏘드는 Blob를 설정하는데 리용된다.

실지로 Blob를 자료기지표에 써넣는 보다 일반적인 방법은 InputStream에서 관계형자료기지관리체계로 자료를 직접 전송하도록 PreparedStatement.setBinaryStream()을 리용하는것이다. 이 방법의 실례를 목록 5-21에 보여주었다.

목록 5-21. Blob를 표에 삽입하는 방법

```
package JavaDB_Bible.ch05.sec04;

import java.io.*;
import java.sql.*;
import javax.sql.*;
```

```
public class BlobSaver {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   public static void main(String args[]) {
      BlobSaver blobber = new BlobSaver();
      blobber.saveImage(1, "PaekDu", "PaekDu.bmp");
   }

   public void saveImage(int imageID, String description,
     String filename) {
      String cmd = "INSERT INTO Photos" +
            "(ImageID, Description, Image) VALUES(?,?,?)";
      File imgFile = new File(filename);

      try {

         Class.forName("com.inet.pool.PoolDriver");
         com.inet.tds.TdsDataSource tds =
               new com.inet.tds.TdsDataSource();
         tds.setServerName("DOLPHIN");
         tds.setDatabaseName("MEMBERS");
         tds.setUser(dbUserName);
         tds.setPassword(dbPassword);

         DataSource ds = tds;
         Connection con = ds.getConnection(dbUserName, dbPassword);

         PreparedStatement pstmt = con.prepareStatement(cmd);

         pstmt.setInt(1, imageID);
         pstmt.setString(2, description);
         pstmt.setBinaryStream(3, new FileInputStream(filename),
                          (int) imgFile.length());

         pstmt.executeUpdate();
         con.close();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      } catch (SQLException e) {
         e.printStackTrace();
```

```
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

목록에서 알수 있는바와 같이 PreparedStatement.setBinaryStream()메쏘드는 setInt()나 setString()와 마찬가지로 쉽게 리용할수 있다.

주의: Blob대면부는 Blob가 포함하고있는 자료가 화상인가 음향클립인가는 전혀 검사하지 않는다.

Clob를 리용한 본문자료의 저장

Clob는 대형자료객체들의 저장과 관리를 위하여 설계되였다는 점에서 Blob와 류사하다. 그러나 Clob의 경우에는 본문객체들을 저장하고 관리한다. Clob와 Blob의 기본차이점은 Clob대면부가 다음과 같은 문자지향적인 접근메쏘드들을 지원한다는것이다.

- public InputStream getAsciiStream()
- public Reader getCharacterStream()
- public String getSubString(long position, int length)

Blob와 마찬가지로 Clob는 Clob의 문자개수와 탐색문자렬에 대한 편위를 돌려주는 length()메쏘드와 position()메쏘드를 가지고있다.

주의: 일반적인 문자렬조종메쏘드들과 달리 getSubString()메쏘드는 0부터가 아니라 1부터 시작하여 문자를 센다. 즉 Clob전체를 문자렬로 돌려주려면 getSubString(1, Clob.length())메쏘드를 리용한다.

ResultSet의 getClob()메쏘드는 ResultSet에서 Clob의 위치자를 검색하는데 리용될수 있으며 PreparedStatement대면부에서의 setClob()메쏘드는 Clob를 설정하는데 리용한다. Blob의 경우와 마찬가지로 자료기지에 Clob를 써넣는 보다 일반적인 방법은 아래의 세가지 setStream()메쏘드들중의 하나를 리용하는것이다.

- setAsciiStream()
- setUnicodeStream()
- setCharacterStream()

이 setStream()메쏘드들중의 하나를 리용하면 InputStream으로부터 관계형자료기지관리체계에로 직접 자료를 전송할수 있다.

목록 5-22에서는 FileReader와 setCharacterStream()메쏘드의 리용을 보여준다.

목록 5-22. FileReader를 리용하여 Clob를 관계형자료기지관리체계에 보관

```
public void saveDocument(int memberID, String title, String filename){
  String cmd = "INSERT INTO Documents " +
    "(MemberID, Title, Document) VALUES (?,?,?)";
  File doc = new File(filename);

  System.out.println(filename + " - " + doc.length());
  try {
    Class.forName("com.inet.pool.PoolDriver");
    com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
    tds.setServerName("DOLPHIN");
    tds.setDatabaseName("MEMBERS");
    tds.setUser(dbUserName);
    tds.setPassword(dbPassword);

    DataSource ds = tds;
    Connection con = ds.getConnection(dbUserName,dbPassword);
    PreparedStatement pstmt = con.prepareStatement(cmd);
    pstmt.setInt(1, memberID);
    pstmt.setString(2, title);
    pstmt.setCharacterStream(3, new FileReader(doc), (int)doc.length());
    pstmt.executeUpdate();
    con.close();
  }catch(ClassNotFoundException e){
    e.printStackTrace();
  }catch(SQLException e){
    e.printStackTrace();
  }catch(FileNotFoundException e){
    e.printStackTrace();
  }
}
```

5.4.2. 열람기로부터 화상과 문서의 올리적재

웨브프로그람들에 대한 일반적인 요구는 인터네트를 통하여 의뢰기로부터 화상과 문서들을 올리적재하는것이다. HTML폼을 리용한 파일의 올리적재는 HTML표준의 일부분이며 주요 열람기들에서 모두 지원하고있는 기능이다.

HTML파일의 올리적재는 다목적인터네트우편확장(Multipurpose Internet Mail Extensions: MIME)표준에 의하여 정의된 여러부분(multipart)통보문형식을 리용하며 폼의 매 마당을 별도의 MIME부분으로 보낸다. HTML올리적재폼을 작성하는데서 주의해야 할 기본 조항은 다음과 같다.

- FORM의 "method"속성을 "post"로 설정한다.
- "enctype=multipart/for-data"속성을 FORM요소에 추가한다.
- "file"형을 가진 INPUT요소가 올리적재하는 파일을 명시하는데 리용된다.

폼이 우와 같이 설정되면 열람기는 사용자가 올리적재할 파일을 선택할수 있게 하는 파일선택콘트롤을 창조한다. 목록 5-23에서는 간단한 HTML올리적재폼의 실례를 보여준다.

목록 5-23. HTML파일적재폼

```
<HTML>
<BODY>
  <FORM action="servlet/JavaDB_Bible.ch05.sec04.BlobUploadServlet"
   enctype="multipart/form-data" method="post">
  <INPUT type="hidden" name="ID" value="1">
  <TABLE border="1">
   <TR>
     <TD align="center">
     Filename:
     <INPUT type="file" name="submit-file" size="40">
     </TD>
   </TR>
   <TR>
     <TD align="center">
     <CENTER>
     <INPUT type="submit" value="Send">
     <INPUT type="reset">
     </CENTER>
     </TD>
   </TR>
  </TABLE>
  </FORM>
</BODY>
</HTML>
```

이 폼에는 그 폼을 창조하는 JSP페지나 써블레트에 의해 설정되는 숨은 MemberID마당이 있다. 이 폼은 또한 파일선택마당도 가지고있다. 목록 5-24에서 보여준 써블레트는 올리적재한 파일을 다시 열람기에 반영하며 따라서 사용자는 올리적재형식을 볼수 있다.

목록 5-24. Blob올리적재 검사써블레트

```
package JavaDB_Bible.ch05.sec04;

import java.io.*;
import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class BlobTestServlet extends HttpServlet{
  public void doPost(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException{
    ServletOutputStream out = response.getOutputStream();
    BufferedInputStream in =
          new BufferedInputStream(request.getInputStream());
    System.out.println(request.getHeader("content-type"));

    int c = -1;
    while((c=in.read())>=0 ) out.write(c);
    out.close();
  }
}
```

만일 사용자가 첫 파일로 GIF파일을 선택하면 자료흐름은 목록 5-25와 같이 나타난다.

목록 5-25. 여러부분자료흐름의 편집된 부분

```
-----------------------------7d514729802d8
Content-Disposition: form-data; name="ID"
1 -----------------------------7d514729802d8
Content-Disposition:          form-data;          name="submit-file";
filename="D:\Image\Child\Doll.GIF"
Content-Type: image/gif GIF87a

. . . . . . . . . . . . . . .

-----------------------------7d514729802d8—
```

여러부분 MIME형식의 자료흐름을 분석하는 한가지 방법은 JavaMail API를 리용하는 것이다. 그러나 보다 간단한 방법은 자료흐름을 사용자자체로 분석하는것이다. 이 방법은 여

러부분 MIME문서분석의 기초를 설명하는 BlobUploadServlet개발에 의해 증명된다.

MIME부분들은 머리부에 정의된 유일한 본문행이며 임의의 MIME내부에 아무런 영향도 주지 않는 경계들에 의해 분리된다. 매 MIME부분은 머리부부분, 빈행, 본체 혹은 자료부(payload)부분으로 이루어진다.

머리부부분에는 본체부분의 내용과 형식을 정의하는 여러개의 머리부들이 있다. 머리부에서는 이름과 값을 두점으로 구분하고 파라메터들은 반두점으로 구분한다. 파라메터들은 HTML에서의 속성과 류사하게 name=value의 쌍으로 되여있다.

MIME경계는 Content-Type머리부에 지적된다. BlobUploadServlet에서 getBoundary()메쏘드는 경계문자렬을 분석하고 CRLF(새행기호)와 두개의 이음표를 앞에 붙인 다음 그 경계를 문자렬로 돌려준다. 자료부 Blob를 검색하는데 쓰이는 read()메쏘드는 이 boundary문자렬을 리용한다.

read()메쏘드는 ServletInputStream으로부터 PushbackInputStream을 작성하고 Stream으로부터 입력문자들을 돌려준다. 만일 경계에 부딪치면 그것을 버리고 경계에 도달했다는것을 지적하는 기발을 돌려준다. 모든 표준문자들이 다 정의옹근수이므로 경계에 맞다들면 -1을 돌려준다.(어떤 경우에는 -2가 돌려지는데 그것은 최종경계이다.)

HTML폼의 마당에 대응하는 매개 부분의 머리부령역은 값 "form-data"를 가진 Content-Disposition머리부를 포함한다. Content-Disposition머리부는 값으로서 HTML폼에 명시된 마당의 이름을 가지는 속성 "name"을 포함한다. 만일 마당의 형이 "file"이면 올리적재되고있는 파일의 이름을 값으로 가지는 "filename"속성을 포함한다.

머리부들은 parseHeader()메쏘드로 분석하는데 이 메쏘드는 머리부파라메터들의 Hashtable를 돌려준다. MemberID와 같은 파라메터들은 파일이름과 다른 머리부에 있으므로 이것들은 파라메터 Hashtable에 합쳐진다.

ServletOutputStream에 대한 머리부정보를 출력하기 위하여 BlobUploadServlet가 작성되여있기때문에 사용자는 ServletOutputStream에 대한 분석결과를 알수 있다. 목록 5-26에 써블레트의 출력결과를 보여준다.

목록 5-26. BlobUploadServlet의 출력결과

```
boundary =
-----------------------------7d5389132030e
Content-Disposition: form-data; name="ID"

Content-Disposition: form-data; name="submit-file";
                     filename="D:\Image\Child\Doll.GIF"

  ID = 1
```

```
filename = D:\Image\Child\Doll.GIF
name = submit-file
Content-Disposition = form-data
Content-Type = image/gif
...saving payload
```

이 써블레트는 Blob의 올리적재를 조종하기 위해 특별히 설계되기는 하였지만 이것을 약간 변경하면 노력을 얼마 들이지 않으면서 Clob를 조종하도록 할수 있다. Content-Type파라메터를 리용하여 올리적재한 파일의 형태를 결정하고 자료를 보관할 때 적당한 JDBC메쏘드들을 선택하면 된다.

만일 적재된 파일이 화상이라면 Content-Type파라메터는 image/pjpeg 혹은 image/gif 등이 될것이다. 이와 류사하게 본문파일을 적재한다면 Content-Type는 자동적으로 text/plain이 되며 MS Word문서라면 application/msword 등이 될것이다.

savePayload()메쏘드는 Blob를 바이트배렬로 분석하고 saveBlob()메쏘드를 리용하여 그것을 자료기지관리체계표에 보관시킨다. saveBlob()메쏘드는 바로 앞 머리부에서 검색되여 자료기지표에 Blob를 보관하기 위하여 리용되는 PreparedStatement에 대한 입력의 하나로서 파라메터 Hashtable에 보관되여있는 회원ID를 리용한다. Blob올리적재써블레트를 목록 5-27에 보여주었다.

목록 5-27. 화상의 올리적재를 위한 BlobUploadServlet

```
package JavaDB_Bible.ch05.sec04;

import java.io.*;
import java.util.*;
import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class BlobUploadServlet extends HttpServlet {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";
   private static final char CR = 13;
   private static final char LF = 10;
   protected String boundary = null;
   protected Hashtable params = new Hashtable();

   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
```

```
    throws ServletException, IOException {
      ServletOutputStream out = response.getOutputStream();
      ServletInputStream in = request.getInputStream();
      BufferedInputStream bin = new BufferedInputStream(in);
      boundary = getBoundary(request.getHeader("content-type"));

      out.println("<html><body><pre>");
      out.println("boundary =\n" + boundary);
      out.println();

      byte[] bytes = new byte[128];
      in.readLine(bytes, 0, bytes.length);
      String line = new String(bytes);
      Hashtable header = null;

      while (in.readLine(bytes, 0, bytes.length) >= 0) {
          line = new String(bytes);
          if (line.startsWith("Content-Disposition:")) {
              out.println(line);
              header = parseHeader(line);
              updateParams(header);
          } else if (line.startsWith("Content-Type:")) {
              params.put("Content-Type",
                   line.substring("Content-Type:".length()).trim());
          } else {
              if (header != null && bytes[0] == 13) {
                  if (header.containsKey("filename")) {
                      displayParams(out);
                      out.println("  ...saving payload");
                      savePayload(params, bin);
                      header = null;
                  } else {
                      String name = (String) header.get("name");
                      String value = getParameter(in).trim();
                      params.put(name, value);
                  }
              }
              if (line.indexOf(boundary) >= 0) out.println(line);
          }
          bytes = new byte[128];
      }
      out.println("</pre></body></html>");
      out.close();
```

```
    }

    private void displayParams(ServletOutputStream out) throws java.io.
            IOException {
        for (Enumeration e = params.keys(); e.hasMoreElements(); ) {
            String key = (String) e.nextElement();
            out.println("  " + key + " = " + params.get(key));
        }
    }

    private void updateParams(Hashtable header) {
        for (Enumeration e = header.keys(); e.hasMoreElements(); ) {
            String key = (String) e.nextElement();
            params.put(key, header.get(key));
        }
    }

    private String getParameter(ServletInputStream in) throws java.io.
            IOException {
        byte[] bytes = new byte[128];
        in.readLine(bytes, 0, bytes.length);
        return new String(bytes);
    }

    private String getBoundary(String contentType) {
        int bStart = contentType.indexOf("boundary=") +
                  "boundary=".length();
        return "" + CR + LF + "--" + contentType.substring(bStart);
    }

    private void savePayload(Hashtable params,
      BufferedInputStream is) throws java.io.IOException {
        int c;
        PushbackInputStream input = new PushbackInputStream(is, 128);
        ByteArrayOutputStream out = new ByteArrayOutputStream();

        while ((c = read(input, boundary)) >= 0) out.write(c);
        int id = Integer.parseInt((String) params.get("ID"));
        saveBlob(id, (String) params.get("filename"), out.toByteArray());
        out.close();
    }

    private int read(PushbackInputStream input, String boundary) throws
```

```
            IOException {
        StringBuffer buffer = new StringBuffer();
        int index = -1;
        int c;

        do {
            c = input.read();
            buffer.append((char) c);
            index++;
        } while ((buffer.length() < boundary.length()) &&
                 (c == boundary.charAt(index)));

        if (c == boundary.charAt(index)) {
            int type = -1;
            if (input.read() == '-') type = -2;
            while (input.read() != LF);
            return type;
        }

        while (index >= 0) {
            input.unread(buffer.charAt(index));
            index--;
        }
        return input.read();
    }

    private Hashtable parseHeader(String line) {
        Hashtable header = new Hashtable();
        String token = null;
        StringTokenizer st = new StringTokenizer(line, ";");

        while (st.hasMoreTokens()) {
            token = ((String) st.nextToken()).trim();
            String key = "";
            String val = "";

            int eq = token.indexOf("=");
            if (eq < 0) eq = token.indexOf(":");
            if (eq > 0) {
                key = token.substring(0, eq).trim();
                val = token.substring(eq + 1);
                val = val.replace('"', ' ');
```

```
                val = val.trim();
                header.put(key, val);
            }
        }
        return header;
    }

    public void saveBlob(int memberID, String description, byte[] out) {
        String cmd = "INSERT INTO Photos " +
                    "(MemberlD,Description,Image) VALUES{?,?,?)";
        System.out.println(cmd);

        try {
            Class.forName("com.inet.pool.PoolDriver");
            com.inet.tds.TdsDataSource tds =
                    new com.inet.tds.TdsDataSource();
            tds.setServerName("JUPITER");
            tds.setDatabaseName("MEMBERS");
            tds.setUser(dbUserName);
            tds.setPassword(dbPassword);

            DataSource ds = tds;
            Connection con = ds.getConnection(dbUserName, dbPassword);

            PreparedStatement pstmt = con.prepareStatement(cmd);
            pstmt.setInt(1, memberID);
            pstmt.setString(2, description);
            pstmt.setBytes(3, out);
            System.out.println(pstmt.executeUpdate());
            con.close();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

5.4.3. 자료기지관리체계로부터 대형객체들을 내리적재하기 위한 써블레트

웨브페지에서 화상이나 기타 대형객체들을 결합하는 편리한 방법은 디스크파일에 대한 련결을 제공하고 OS가 그 파일을 찾도록 하는것이다. 이것은 화상파일들의 개수가 적은 경우에는 효과적인 방법으로 되지만 회원들이 수십 혹은 수십만명을 헤아리는 회원싸

이트에서는 매 회원들이 여러개의 사진들을 파일로 가지고있어야 하므로 검색시간이 중요하게 제기된다. 이것을 해결하는 한가지 방법은 나무구조의 등록부를 설계하고 일정한 론리적방법으로 수백개의 부분등록부들을 정렬시켜 사용자가 정확한 부분등록부에서 빨리 열람할수 있게 하는것이다.

화상파일들을 찾는 훨씬 더 고급하고 매력적인 방법은 자료기지관리체계에 이 작업을 시키는것이다. 객체관계형자료기지관리체계의 큰 우점은 그것들이 바로 이러한 종류의 일을 위해 특별히 설계되였다는것이다.

목록 5-28의 써블레트에서는 자료기지관리체계로부터 화상을 Blob로 검색하고 그것을 ServletOutputStream에 바이트배렬로 써넣는 방법을 보여준다. 이 써블레트는 본문을 Clob로 검색하도록 고칠수도 있다.

경고: HTML이 아닌 자료를 내리적재할 때에는 대응하는 객체의 형태를 정확히 설정하는것이 중요하다. 일부 열람프로그람들에서는 다른 열람기들보다 이에 더 민감하다.

목록 5-28. 대형객체들을 검색하는 써블레트

```
package JavaDB_Bible.ch05.sec04;

import java.io.*;
import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LobServlet extends HttpServlet {
   private String dbUserName = "sa";
   private String dbPassword = "dba";

   protected void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws ServletException,
          IOException {
      ServletOutputStream out = response.getOutputStream();
      String dataType = request.getParameter("type");
      int memberID = Integer.parseInt(request.getParameter("id"));
      if (dataType.equalsIgnoreCase("blob")) {
         response.setContentType("image/jpeg");
         out.write(getBlob(memberID));
      } else if (dataType.equalsIgnoreCase("clob")) {
         response.setContentType("text/html");
         out.write(getClob(memberID));
      }
      out.flush();
      out.close();
   }
```

```
public byte[] getBlob(int memberID) {
    String query = "SELECT Image FROM Photos WHERE MemberID = ?";
    Blob blob = null;
    byte[] bytes = null;
    String description = "";
    try {
        Class.forName("com.inet.pool.PoolDriver");
        com.inet.tds.TdsDataSource tds =
                new com.inet.tds.TdsDataSource();
        tds.setServerName("DOLPHIN");
        tds.setDatabaseName("MEMBERS");
        tds.setUser(dbUserName);
        tds.setPassword(dbPassword);

        DataSource ds = tds;
        Connection con = ds.getConnection(dbUserName, dbPassword);

        PreparedStatement pstmt = con.prepareStatement(query);
        pstmt.setInt(1, memberID);
        ResultSet rs = pstmt.executeQuery();
        ResultSetMetaData md = rs.getMetaData();
        while (rs.next()) {
            blob = rs.getBlob(1);
        }
        bytes = blob.getBytes(1, (int) (blob.length()));
        con.close();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return bytes;
}

public byte[] getClob(int memberID) {
    String query = "SELECT Document FROM Documents WHERE MemberID = ?";
    Clob clob = null;
    String text = null;

    try {
        Class.forName("com.inet.pool.PoolDriver");
        com.inet.tds.TdsDataSource tds =
                new com.inet.tds.TdsDataSource();
        tds.setServerName("DOLPHIN");
        tds.setDatabaseName("MEMBERS");
```

```
            tds.setUser(dbUserName);
            tds.setPassword(dbPassword);

            DataSource ds = tds;
            Connection con = ds.getConnection(dbUserName, dbPassword);

            PreparedStatement pstmt = con.prepareStatement(query);
            pstmt.setInt(1, memberID);
            ResultSet rs = pstmt.executeQuery();
            ResultSetMetaData md = rs.getMetaData();
            while (rs.next()) {
                clob = rs.getClob(1);
            }
            text = clob.getSubString(1, ((int) clob.length()));
            con.close();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        byte[] bytes = null;
        if (text != null) bytes = text.getBytes();
        return bytes;
    }
}
```

LobServlet는 Blob로 보관된 화상들과 Clob로 보관된 본문이나 HTML을 결합하여 웨브페지를 구동시키는데 리용할수 있다. 목록 5-29는 간단한 설명을 보여준다.

목록 5-29. 프레임으로 Blob와 Clob에 기초한 웨브페지를 창조한다.

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=big5">
<title>Blob Clob Download</title>
</head>

<frameset rows="50%,*">
   <frame
src="http://localhost:8080/MySample/servlet/JavaDB_Bible.ch05.sec04.L
obServlet?type=blob&id=1">
   <frame
src="http://localhost:8080/MySample/servlet/JavaDB_Bible.ch05.sec04.L
obServlet?type=clob&id=1">
</frameset>
```

```
<body>
</body>

</html>
```

그림 5-11에서 그림은 MemberID=1로 저장된 JPEG화상이며 본문은 HTML폼에서 같은 ID를 가지고 Clob로 저장된것이다. HTML프레임설정은 단순히 페지를 형식화하기 위하여 필요하며 그 내용은 전적으로 자료기지에 의하여 구동된다.

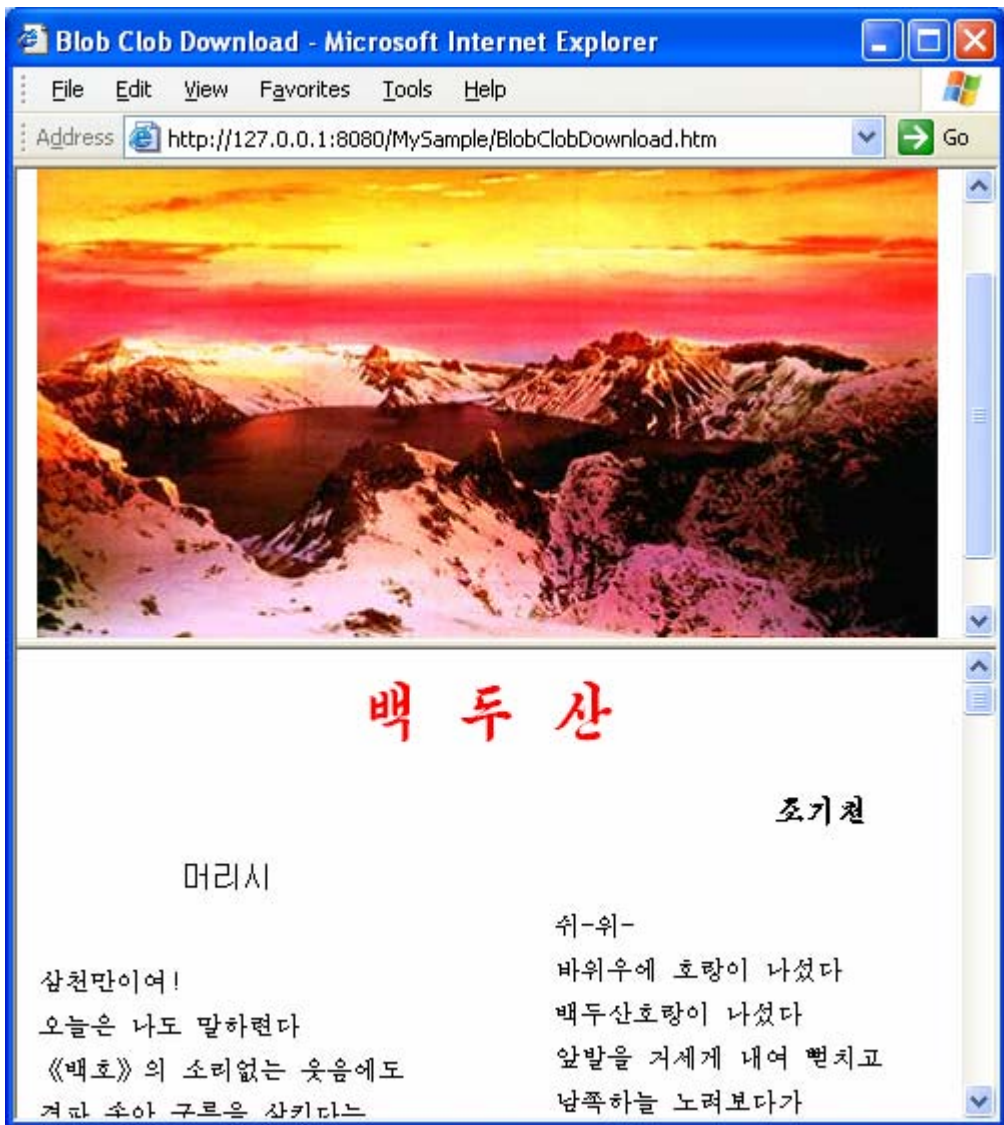



그림 5-11. 프레임을 리용한 Blob와 Clob에 기초한 웨브페지

제5절. JSP, XSL, 흘리기가능한 ResultSet를 리용한 자료의 현시

JDBC 2.0 API는 사용자가 유표를 임의의 방향으로 혹은 특정한 행으로 이동할수 있도록 ResultSet를 흘리기가능하게 정의하도록 하였다. 이 절에서는 흘리기가능한 ResultSet를 리용한 싸이트탐색엔진의 작성, ResultSet로부터 XML문서작성, 웨브페지작성을 위해 XML문서에 XSL양식일람의 적용, 자료기지레코드갱신을 위한 HTML폼과 갱신가능한 ResultSet의 리용에 대하여 학습한다.

5.5.1. 흘리기가능한 ResultSet

java.sql.Statement객체가 돌려주는 ResultSet의 류형은 Statement가 Connection.createStatement메쏘드에 의하여 창조될 때 정의된다.

흘리기가능한 ResultSet의 창조

흘리기 및 갱신가능한 ResultSet를 창조할수 있는 Connection.createStatement메쏘드형식은 다음과 같다.

```
public Statement createStatement(int rsType, int rsConcurrency)
```

여기서 첫번째 인수 rsType는 ResultSet객체의 류형을 지적하기 위한 상수로서 다음의 3가지 상수들중 하나가 되여야 한다.

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

만일 흘리기가능한 ResultSets객체를 만들고 싶다면 ResultSet객체의 류형을 TYPE_SCROLL_INSENSITIVE나 TYPE_SCROLL_SENSITIVE로 지정해야 한다. TYPE_SCROLL_INSENSITIVE로 정의된 ResultSet는 열려져있는 동안에는 변경된 내용을 자료기지에 반영하지 않는다. 다시 말하여 닫겨진 다음에야 변경된 내용이 반영된다. TYPE_SCROLL_SENSITIVE로 정의된 ResultSet는 변경된 내용이 즉시에 반영된다. 물론 ResultSet의 류형에 관계없이 ResultSet를 닫았다가 다시 열면 변경된 내용을 항상 볼수 있다.

만일 TYPE_FORWARD_ONLY로 지정하면 흘리기할수 없는 ResultSet를 얻게 되는데 여기에서는 유표를 앞으로만 이동시킬수 있다. 또한 두번째 인수를 CONCUR_READ_ONLY라고 지정하면 아무런 인수도 없이 만든 ResultSet와 꼭 같은 기정의 ResultSet를 얻게 된다.

두번째 인수는 ResultSet가 읽기전용인가 혹은 갱신할수 있는가를 규정하는 상수로서 다음 두가지 상수들중의 하나가 되여야 한다.

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

주의: ResultSet의 류형을 지적하면 반드시 그것이 읽기전용인가 혹은 갱신가능한가를 반드시 지정해야 한다.

아래에서와 같이 ResultSet.getType()메쏘드를 리용하면 현재 리용하고있는 ResultSet의 류형을 검사할수 있다.

```
if(rs.getType()==ResultSet.TYPE_FORWARD_ONLY)
   System.out.println("FORWARD_ONLY");
else
   System.out.println("SCROLLABLE");
```

흘리기가능한 ResultSet에서 유표이동

일단 흘리기가능한 ResultSet객체를 가지면 다음의 메쏘드들을 리용하여 유표를 앞뒤방향으로 다 이동할수 있다.

- ResultSet.next() : ResultSet에서 유표를 다음 행으로 이동시킨다.
- ResultSet.previous() : ResultSet에서 유표를 한행 앞으로 이동시킨다.

유표가 ResultSet의 밖으로 벗어나면 이 메쏘드들은 둘다 "false"를 돌려준다. 따라서 while순환에서 이 메쏘드들을 쉽게 리용할수 있다.

다음의 메쏘드들은 직접 지적된 행으로 유표를 이동시킨다.

- first() : 유표를 맨 첫행으로 이동시킨다.
- last() : 유표를 마지막행으로 이동시킨다.
- beforeFirst() : 유표를 첫 행의 바로 앞으로 이동시킨다.
- afterLast() : 유표를 마지막행의 바로 뒤로 이동시킨다.
- absolute(int rowNumber) : 유표를 지적한 행으로 이동시킨다.
- relative(int rowNumber) : 유표를 지적한 행수만큼 이동시킨다.

absolute(int rowNumber)메쏘드는 유표를 인수에서 지적한 행번호로 이동시킨다. 만일 그 수가 정수이면 맨 첫행부터 시작하여 주어진 행번호로 이동시키고 그 수가 부수이면 뒤에서부터 주어진 행번호만큼 이동시킨다. 다시 말하여 absolute(1)은 유표를 첫 행으로 이동시키고 absolute(-1)은 마지막행으로 이동시킨다.

relative(int rowNumber)메쏘드는 현재 유표가 위치한 행에서 얼마만한 행수만큼 이동해야 하는가와 이동할 방향을 지적할수 있게 한다. 정수이면 유표를 주어진 행수만큼 뒤로 이동시키고 부수이면 앞으로 이동시킨다.

탐색페지작성에 흘리기가능한 ResultSet의 리용

앞에서 우리는 써블레트와 JSP페지를 리용하여 HTML폼들을 조종하고 폼자료, 화상, 문서들을 자료기지에 보관하는 방법을 보았다. 이 절에서는 자료기지에서 자료들을 탐색하고 그것을 웨브페지에 보여주는데 중심을 둔다.

웨브싸이트는 회원들이 탐색조건을 입력하고 형식화된 ResultSet를 흘리기할수 있게 하는 탐색능력을 가진다. 사용자가 탐색결과중에서 어느 하나를 찰칵하면 자료기지에서 그 항목에 대한 보다 많은 정보를 불러내여 보여주는 상세페지가 현시된다.

탐색의 출발점은 역시 사용자가 자기의 탐색기준을 설정할수 있게 하는 폼이다. 간단한 탐색폼을 그림 5-12에 보여주었다. 이 폼이 정의하는 탐색조건은 JSP페지와 JavaBean을 리용하여 수집된다. 탐색조건은 목록 5-30에 보여준 폼의 SQL저장수속에 입력파라메터로 넘겨진다.

목록 5-30. 대응하는 자료기지항목들을 돌려주는 SQL저장수속

```
CREATE PROCEDURE SEARCH
  @BODY VARCHAR(50), @ZIP VARCHAR(10),
  @MAKE VARCHAR(50), @MODEL VARCHAR(50),
  @ENGINE VARCHAR(50), @TRANSMISSION VARCHAR(50),
  @PRICE INT, @YEAR1 INT, @YEAR2 INT
  AS SELECT TOP 50 *
    FROM VEHICLES
    WHERE BODY LIKE @BODY AND
      ZIP LIKE @ZIP AND
      MAKE LIKE @MAKE AND
      MODEL LIKE @MODEL AND
      ENGINE LIKE @ENGINE AND
      TRANSMISSION LIKE @TRANSMISSION AND
      PRICE <= @PRICE AND
      YEAR BETWEEN @YEAR1 AND @YEAR2;
```

저장수속은 통용문자를 리용할수 있는 LIKE비교연산자를 리용한다. 이렇게 하면 유연하게 자료기지를 탐색할수 있다. 아래의 HTML부분에서는 Any라는 추가선택항목이 선택될 때 통용문자 "%"를 돌려주도록 SELECT요소를 정의하는 방법을 보여주고있다.

```
<TR>
  <TD>Make: </TD>
  <TD>
     <SELECT name=Make size=1>
        <OPTION VALUE="%" SELECTED>Any</OPTION>
        <OPTION VALUE="Acura">Acura</OPTION>
        <OPTION VALUE="Audi">Audi</OPTION>
        <OPTION VALUE="BMW">BMW</OPTION>
     </SELECT>
  </TD>
</TR>
```

목록 5-30의 저장수속에서 TOP 50이라는 문절의 리용에 주의하기 바란다. 그것은 돌려지는 탐색결과의 수를 제한하는데 쓰인다. 일반적인 조건에서 큰 자료기지가 탐색되는 경우 탐색결과가 많아지고 ResultSet가 큰 기억공간을 차지하게 된다.

만일 ResultSet를 JavaBean에 돌려주고 그것을 흘리기할수 있게 하면 사용자가 흘리기할수 있는 페지당 5개의 자료기지항목이 나타나도록 함으로써 페지작성이 쉬워질것이다. 탐색결과를 기본열쇠로 순서화하고 뒤로 잇달리는 Resultset가 보다 높은 기본열쇠값을 가지도록 지적하는것에 의해 본래의 탐색조건에 기초한 50개 결과들의 블로크들을 요구하는 추가선택항목을 항상 사용자들에게 제공할수 있다.

그림 5-12에서 보여준 탐색폼은 JavaBean을 리용하여 질문을 처리하고 그 결과들을 돌려주는 간단한 JSP페지 ProcessSearchForm.jsp를 호출한다. 목록 5-31에서 보여주고있는 이 페지는 SearchFormBean.getMatches()메쏘드를 호출하여 bean을 적재하고 속성들을 설정하며 질문을 수행한다. 다음 사용자는 검색결과를 현시하는 SearchFormResultsPage.jsp에로 이동한다.

그림 5-12. 검색폼

목록 5-31. 질문작성 JavaBean을 적재하는 JSP페지(ProcessSearchForm.jsp)

```
<%@ page language="java"%>
<jsp:useBean id="SearchFormBean"
   class="JavaDB_Bible.ch05.sec05.SearchFormBean"
   scope="session"/>
<jsp:setProperty name="SearchFormBean" property="*"/>
<%SearchFormBean.getMatches();%>
<jsp:forward page="SearchFormResultsPage.jsp"/>
```

목록 5-32에서는 SearchFormBean자체를 보여준다.

목록 5-32. JSP페지에서 자료기지질문을 다루기 위한 JavaBean

```
package JavaDB_Bible.ch05.sec05;

import java.sql.*;
import javax.sql.*;

public class SearchFormBean extends java.lang.Object {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   protected int price;
   protected int year;
   protected String id;
   protected String make;
   protected String model;
   protected String color;
   protected String body;
   protected String engine;
   protected String transmission;
   protected String zip;

   protected int index = 0;
   protected int pageSize = 5;
   protected int rowCount = 0;

   protected ResultSet rs = null;

   public SearchFormBean() {
   }

   public void setYear(int year) {
      this.year = year;
   }

   public void setMake(String make) {
      this.make = make;
   }

   public void setZip(String zip) {
      this.zip = zip;
   }
```

```
public void setModel(String model) {
   this.model = model;
}

public void setColor(String color) {
   this.color = color;
}

public void setBody(String body) {
   this.body = body;
}

public void setEngine(String engine) {
   this.engine = engine;
}

public void setTransmission(String transmission) {
   this.transmission = transmission;
}

public void setPrice(int price) {
   this.price = price;
}

public String getId() {
   return id;
}

public String getMake() {
   return make;
}

public String getZip() {
   return zip;
}

public String getModel() {
   return model;
}

public String getColor() {
   return color;
```

```
}

public String getBody() {
   return body;
}

public String getEngine() {
   return engine;
}

public String getTransmission() {
   return transmission;
}

public int getPrice() {
   return price;
}

public int getYear() {
   return year;
}

public int getIndex() {
   return index;
}

public int getRowCount() {
   return rowCount;
}

public String getPage() {
   return "Page " + (index / pageSize + 1) + " of " +
        (rowCount / pageSize + 1);
}

public boolean pageForward() {
   boolean validRow = false;
   if (index < 0 || index + pageSize > rowCount) {
      index = 0;
   } else {
      index += pageSize;
   }
```

```
        try {
            validRow = rs.absolute(index + 1);
        } catch (SQLException e) {
            System.err.println(e.getMessage());
        }
        return validRow;
    }

    public boolean pageBack() {
        boolean validRow = false;
        if (index < pageSize) {
            index = rowCount / pageSize * pageSize;
        } else if (index >= pageSize) {
            index -= pageSize;
        }
        try {
            validRow = rs.absolute(index);
        } catch (SQLException e) {
            System.err.println(e.getMessage());
        }
        return validRow;
    }

    public boolean selectRow(int row) {
        boolean validRow = false;
        try {
            validRow = rs.absolute(index + 1);
            if (validRow) {
                if (row > 0) validRow = rs.relative(row);
                if (rs.getRow() < 0) validRow = false;

                if (validRow) {
                    id = rs.getString("ID");
                    year = rs.getInt("year");
                    make = rs.getString("make");
                    zip = rs.getString("zip");
                    model = rs.getString("model");
                    body = rs.getString("body");
                    engine = rs.getString("engine");
                    transmission = rs.getString("transmission");
                    price = rs.getInt("price");
                }
```

```
        }
    } catch (SQLException e) {
        System.err.println(e.getMessage());
    }
    return validRow;
}

/*
getMatches메쏘드는 SEARCH저장수속을 리용한다
CREATE PROCEDURE SEARCH @BODY VARCHAR(50),
  @ZIP VARCHAR(10), @MAKE VARCHAR(50),
  @MODEL VARCHAR(50), @ENGINE VARCHAR(50),
  @TRANSMISSION VARCHAR{50), @PRICE INT, @YEAR1 INT,
  @YEAR2 INT AS SELECT TOP 50 *
  FROM VEHICLES
  WHERE BODY LIKE @BODY AND
   ZIP LIKE @ZIP AND MAKE LIKE @MAKE AND
   MODEL LIKE @MODEL AND ENGINE LIKE @ENGINE AND
   TRANSMISSION LIKE @TRANSMISSION AND
   PRICE <= @PRICE AND YEAR >= @YEAR;
*/

public int getMatches() {
    try {
        Class.forName("com.inet.pool.PoolDriver");
        com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
        tds.setServerName("DOLPHIN");
        tds.setDatabaseName("MEMBERS");
        tds.setUser(dbUserName);
        tds.setPassword(dbPassword);

        DataSource ds = tds;
        Connection con = ds.getConnection(dbUserName, dbPassword);

        // CallableStatement에 대한 자유형식의 본문마당들을 정돈
        if (model == null) model = "%";
        if (zip == null) zip = "%";

        CallableStatement cs = con.prepareCall(
              "{call SEARCH (?,?,?,?,?,?,?,?)}");
        cs.setString(1, body);
        cs.setString(2, zip);
```

```
        cs.setString(3, make);
        cs.setString(4, model);
        cs.setString(5, engine);
        cs.setString(6, transmission);
        cs.setInt(7, price);
        cs.setInt(8, year);

        rs = cs.executeQuery();
        rs.last();
        rowCount = rs.getRow();
        con.close();
      } catch (ClassNotFoundException e1) {
        System.err.println(e1.getMessage());
      } catch (SQLException e2) {
        System.err.println(e2.getMessage());
      }
      return rowCount;
    }
  }
```

이 JavaBean은 속성들에 대한 표준적인 취득자메쏘드와 설정자메쏘드외에도 흘리기 가능한 ResultSet에서 자료를 탐색하고 열람할수 있는 많은 메쏘드들을 가지고있다. 그 메쏘드들은 다음과 같다.

- getMatches(): 이 메쏘드는 SQL저장수속의 파라메터들을 설정하고 흘리기 가능한 ResultSet를 얻기 위하여 그 저장수속을 호출한다. 다음 ResultSet의 마지막 행에 가서 getRow()로 행번호를 얻어 탐색결과의 총개수를 얻는다. 다음 Resultset는 다른 메쏘드들에 의한 접근때문에 JavaBean에 저장된다.
- selectRow(int row): 이 메쏘드는 유표를 선택한 행으로 이동시킨다. row 인수는 현시된 JSP페지안에서의 행번호를 가리킨다. selectRow()메쏘드는 우선 ResultSet.absolute(index)메쏘드를 써서 유표를 현시된 JSP페지의 첫 행에 대응하는 행으로 이동시킨다. 옹근수 index는 현재 JSP페지의 시작에 대응하는 행에 대한 편위를 제공한다. 다음 ResultSet.relative(row)를 리용하여 현시된 JSP페지의 해당한 행으로 이동한다. 드디여 ResultSet로부터 적당한 자료들을 얻어 JavaBean의 속성들을 설정한다.
- pageForward(): pageForward()메쏘드는 행색인을 다음 페지의 꼭대기에 대응하는 행으로 이동시킨다.
- pageBack(): pageBack()메쏘드는 행색인을 이전 페지의 첫 부분에 대응하

는 행으로 이동시킨다.

- getRowCount(): getRowCount()메쏘드는 행개수를 돌려준다.
- getPage(): getPage()메쏘드는 "Page 1 of n"과 같은 형식으로 현재 페지번호에 대한 문자렬표현을 돌려준다.

목록 5-32에서 약간 복잡한 론리는 유표를 이동시키는 부분에 있다. 절대행번호는 1로부터 시작하며 상대행번호는 령이 될수 없다는데 주의해야 한다. 탐색결과를 현시하는데 리용되는 JSP페지는 bean의 구체례를 만들고 질문을 수행하는 JSP페지와 분리되여 유지된다. 그것은 페지단추들을 포함하는 두가지 HTML폼요소들을 포함하고있으며 ResultSet를 통한 열람을 다루기 위하여 다른 JSP페지들을 호출한다. 현시페지를 위한 코드를 목록 5-33에 보여준다.

목록 5-33. 탐색결과페지 JSP

```
<%@ page language="java"%>
<jsp:useBean id="SearchFormBean"
class="JavaDB_Bible.ch05.sec05.SearchFormBean" scope="session"/>
<html>
<head>
<title>Summary</title>
</head>
<body bgcolor="#ffffff">
<BASEFONT FACE="Arial">
<TABLE BORDER="2">
  <TR BGCOLOR="#E0E0E0">
  <TD COLSPAN="2">

  <!-- header -->
  <TABLE WIDTH=100%>
  <TR BGCOLOR="#E0E0E0">
    <TD>
    Found
      <%=SearchFormBean.getRowCount()%> vehicles matching query.
    </TD>
    <TD ALIGN="RIGHT">
    Page
    <%=SearchFormBean.getPage()%>
    </TD>
  </TR>
  </TABLE>
  </TD>
  </TR>
```

```
<!-- results -->
<%
if(SearchFormBean.getRowCount()>0){
   for(int i=0;i<3;i++){
      if(SearchFormBean.selectRow(i)){
%>

<TR>
<TD>
   <A HREF="GetDetailPage.jsp?memberId=<%=SearchFormBean.getId()%>">
   <img src = "http://127.0.0.1:8080/MySample/servlet/
            JavaDB_Bible.ch05.sec04.LobServlet?type=blob&
            id=<%=(i+5230001)%>&description=Thumbnail">
   </A>
</TD>
<TD>
   <TABLE CELLPADDING=4 width=100%>
   <TR>
      <TD>
      <%=SearchFormBean.getYear()%>,
      <%=SearchFormBean.getMake()%>,
      <%=SearchFormBean.getModel()%>,
      <%=SearchFormBean.getBody()%>,
      <%=SearchFormBean.getEngine()%>,
      <%=SearchFormBean.getTransmission()%>,
      Asking $<%=SearchFormBean.getPrice()%>,
      Location ( zip code ): <%=SearchFormBean.getZip()%>
      </TD>
   </TR>
   </TABLE>
</TD>
</TR>

<!-- footer -->
<TR BGCOLOR="#E0E0E0">
<TD COLSPAN="2">
   <TABLE WIDTH=100%>
   <TR BGCOLOR="#E0E0E0">
      <TD WIDTH="60%">
      </TD>
      <TD>
      <form METHOD="POST" ACTION="SearchFormPageBack.jsp"
         target= "_self">
```

```
        <input type="submit" value="Prev Page">
        </form>
        </TD>
        <TD>
        <form METHOD="POST" ACTION="SearchFormPageForward.jsp"
          target= "_self">
        <input type="submit" value="Next Page">
        </form>
        </TD>
        <TD ALIGN="RIGHT">
        </TD>
      </TR>
      </TABLE>
    </TD>
    </TR>
</TABLE>
</body>
</html>
```

<Prev Page>단추와 <Next Page>단추를 지원하는 JSP페지들도 기본적인 JSP페지 ProcessSearchForm만큼 간단하다.

SearchFormPageForward.jsp는 페지크기만큼 페지색인변수를 증가시키기 위하여 SearchFormBean의 PageForward()메쏘드를 호출한다.

```
<%@ page language="java"%>
<jsp:useBean id="SearchFormBean"
           class="JavaDB_Bible.ch05.sec05.SearchFormBean"
   scope="session"/>
<%=SearchFormBean.pageForward()%>
<jsp:forward page="SearchFormResultsPage.jsp"/>
```

SearchFormPageBack.jsp는 페지크기만큼 페지색인변수를 감소시키기 위하여 SearchFormBean.pageBack()메쏘드를 호출한다.

```
<%@ page language="java"%>
<jsp:useBean id= "SearchFormBean"
           class="JavaDB_Bible.ch05.sec05.SearchFormBean"
   scope="session"/>
<%=SearchFormBean.pageBack()%>
<jsp:forward page="SearchFormResultsPage.jsp"/>
```

SearchFormResultsPage.jsp가 창조하는 페지를 그림 5-13에서 보여주었다.

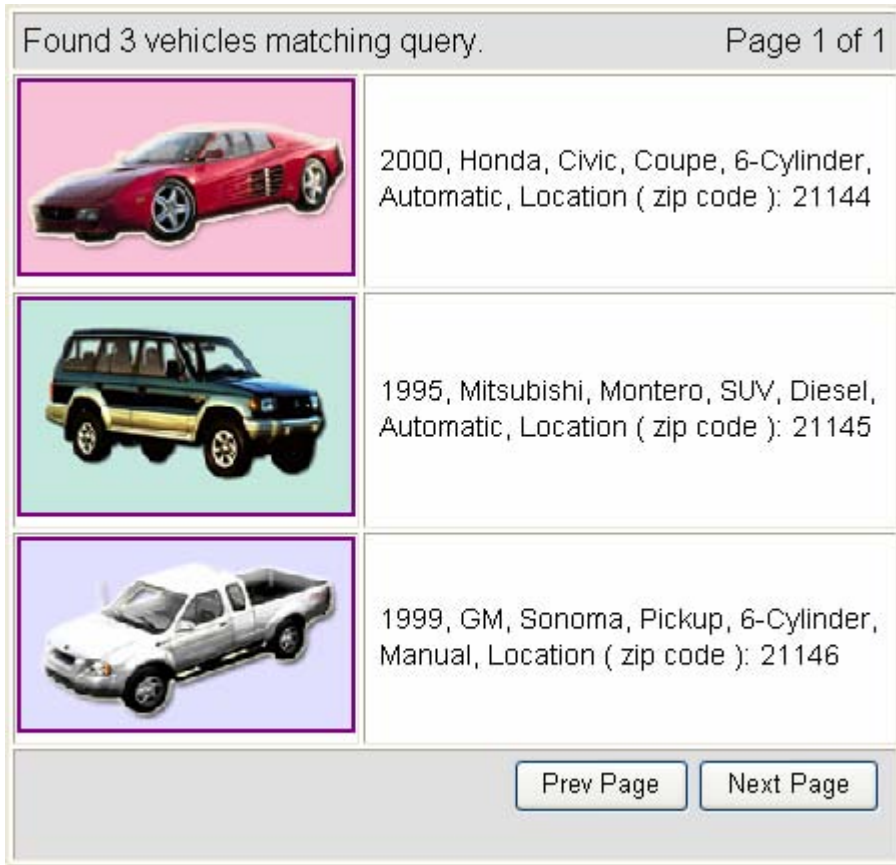



그림 5-13. 탐색결과페지

목록 5-33에서 알수 있는것처럼 화상들은 자료기지의 Photos표에서 Blob들을 돌려주는 써블레트로부터 얻어진다. 이 써블레트는 목록 5-28의 LobServlet실례로부터 유도된것이다. 탐색결과페지의 자그마한 화상들을 클릭하면 그 자동차에 대한 상세페지로 넘어가게 된다.

5.5.2. XSL을 리용한 웨브페지작성

지금까지의 실례에서는 웨브페지의 형식화를 관리하는 JSP를 리용하여 웨브페지를 작성하는 방법을 보여주었다. 이 방법의 부족점은 현시형식을 변경하려면 JSP의 리용법을 잘 알아야 한다는것이다.

자료기지구동형 웨브페지의 형식화를 관리하는 또 한가지 방법은 XML자료를

HTML로 변환하기 위한 XSL양식일람의 리용이다. XSLT에 대한 간단한 변화는 같은 XML페지를 완전히 서로 다른 HTML페지로 변환한다.

XSLT는 어떻게 일하는가

Extensible Stylesheet Language(XSL)은 사용자에게 XML문서들을 한가지 형식으로부터 다른 형식으로 변환하는 방법을 제공한다. 실천적으로 이것은 자료기지로부터 정보를 기본적인 내용지향의 XML로 검색하고 XSL양식일람을 리용하여 그것을 사람이 쉽게 읽을수 있는 문서로 변환할수 있다는것을 의미한다.

XSL은 실제상 두개의 기본성분들 즉 변환언어와 형식언어를 결합하고있다. 이것들은 다 XML로부터 파생된 언어들이다. XSL변환언어(XSL Transformation Language)는 XML문서를 다른 XML문서로 변환하는 규칙들을 정의하는데 리용되며 형식화구성요소는 출력하는 내용의 형식화를 다룬다.

HTML을 생성하는 시점에서 보면 중요한 성분이 XSL변환(XSL Transfomation: XSLT)언어이다. 변환을 수행하기 위하여 XSLT처리기는 XML문서와 XSLT양식일람을 둘다 읽고 새로운 XML문서를 내보낸다.

Java에서 XSL변환의 적용은 목록 5-34의 실례에서 알수 있는바와 같이 아주 간단하다. xalan서고의 메쏘드들이 대부분의 작업을 진행한다.

목록 5-34. XSL변환의 적용

```
import org.xml.sax.SAXException;
import org.apache.xalan.xslt.XSLTProcessorFactory;
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;
import org.apache.xalan.xslt.XSLTProcessor;

// HTML페지를 창조하기 위하여 xml문서에 양식일람을 적용하는 실레코드
public class SimpleXSLTransform{
   public static void main(String[] args)
      throws org.xml.sax.SAXException{
      XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
      processor.process(new XSLTInputSource("MemberInfo.xml"),
         new XSLTInputSource("Memberlnfo.xsl"),
         new XSLTResultTarget("MemberInfo.html"));
   }
}
```

XSL양식일람들은 봉사기측에서 리용하든가 혹은 의뢰기측에서 리용할수 있다. 실천적으로는 봉사기측 변환이 더 잘 동작한다. 그것은 각이한 열람기들이 명세의 각이한 부분모임을 실현하기때문에 결과들을 예측할수 없다. 한편 봉사기에서 XSL변환을 리용할 때의 기본약점은 그것들이 자원집중적이기 쉽다는것이다. 일부 변환서고들은 다른것들보다 변환을 훨씬 더 빠르게 수행하기때문에 서로 다른 XSL변환서고들을 가지고 실험해볼 필요가 있다.

자료기지로부터 XML문서로 자료검색

물론 결과자료를 변환하기전에 ResultSet를 얻어야 하며 그 다음에야 그것을 XML로 변환해야 한다. 그림 5-13에서 본 탐색결과페지를 작성하는 자료모임은 한개 표에서 얻은것이다. 보다 구체적인 페지를 작성하기 위해서는 여러 표들에 있는 정보들을 결합해야 한다.

상세페지에서는 매 자동차에 대한 기본정보를 가지고있는 Vehicle표와 부속물들과 선택적인 부품들에 대한 정보를 포함하고있는 Options표에 접근하게 된다.

Options표의 설계에서 중요한 측면은 이 표가 HTML폼에서 "Other"라고 표시된 간단한 본문항목들은 물론 Yes/No값을 가진 검사칸 선택을 나타내는 수많은 정보들을 포함하고 있다는것이다. 폼에 입력한 자료가 자료기지표에 보관되면 이 HTML폼에 입력된 모든 자료들은 LIST라는 표식의 렬에 결합된다. 이런 방법으로 표를 설계하면 속성들의 본문개요창조로 인한 부가적처리없이 특정한 Yes/No속성들에 대한 검색을 쉽게 할수 있다.

목록 5-35의 SQL문들에서 보는바와 같이 상세페지에 대한 ResultSet는 속성표들의 LIST렬들에 기초하고있다. 이 렬들은 Vehicle표의 개별적인 렬들의 정보와 결합된다. 목록 5-35에서는 저장수속 GET_DETAIL_PAGE를 보여주고있다. 이 저장수속은 XML문서를 작성하는데 리용되는 JavaBean에서 호출하게 된다.

목록 5-35. 상세페지에 대한 저장수속

```
CREATE PROCEDURE GET_DETAIL_PAGE
   @id int
   AS SELECT v.*, o.list Options
   FROM Vehicles v, Options o
   WHERE o.VehicleID = v.VehicleID AND v.VehicleID = @id;
```

목록 5-36에서는 목록 5-35의 저장수속을 호출하고 얻어진 ResultSet를 XML로 형식화하는 JavaBean을 보여주고있다. ResultSetMetaData객체는 XML요소들에 대한 태그이름으로 리용되는 렬이름을 얻기 위하여 리용한다.

목록 5-36. ResultSet를 XML로 돌려주는 JavaBean

```
package JavaDB_Bible.ch05.sec05;

import java.io.*;
import java.sql.*;
import javax.sql.*;

public class DetailPageXMLBean {
    protected static String dbUserName = "sa";
    protected static String dbPassword = "dba";
    protected String xmlHeader = "<?xml version=\"1.0\" encoding=\"big5\"?>";

    protected int id;

    public DetailPageXMLBean() {
    }

    public static void main(String args[]) {
        File f = new File("Detail.xml");
        int id = 1001;

        DetailPageXMLBean xmlBean = new DetailPageXMLBean();
        xmlBean.setId(id);
        try {
            FileOutputStream fos = new FileOutputStream(f);
            fos.write(xmlBean.getVehicleData());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getXmlString() {
        String xml = new String(getVehicleData());
        return xml.trim();
    }

    public byte[] getVehicleData() {
        String rootTag = "VehicleData";
```

```
ByteArrayOutputStream os = new ByteArrayOutputStream();

try {

    Class.forName("com.inet.pool.PoolDriver");
    com.inet.tds.TdsDataSource tds =
          new com.inet.tds.TdsDataSource();
    tds.setServerName("DOLPHIN");
    tds.setDatabaseName("MEMBERS");
    tds.setUser(dbUserName);
    tds.setPassword(dbPassword);

    DataSource ds = tds;
    Connection con = ds.getConnection(dbUserName, dbPassword);

    Statement stmt = con.createStatement();
    CallableStatement cs =
          con.prepareCall("{call GET_DETAIL_PAGE(?)}");

    cs.setInt(1, id);
    ResultSet rs = cs.executeQuery();
    ResultSetMetaData md = rs.getMetaData();

    os.write(xmlHeader.getBytes());
    os.write(("\n").getBytes());
    os.write(("<" + rootTag + ">" + "\n").getBytes());

    String xml = "";
    int columns = md.getColumnCount();
    rs.next();

    for (int i = 1; i <= columns; i++) {
        if (md.getColumnType(i) == Types.VARCHAR) {
            xml = "<" + md.getColumnLabel(i) + ">" +
                  rs.getString(i) + "</" +
                  md.getColumnLabel(i) + ">"+"\n";
            os.write(xml.getBytes());
        } else if (md.getColumnType(i) == Types.INTEGER) {
            xml = "<" + md.getColumnLabel(i) + ">" +
                  rs.getInt(i) + "</" +
                  md.getColumnLabel(i) + ">"+"\n";
            os.write(xml.getBytes());
```

```
            }
        }
        os.write(("</" + rootTag + ">").getBytes());
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return os.toByteArray();
  }
}
```

이것은 XML을 생성하는 하나의 간단한 방법으로서 DOM객체의 구축과 직렬화로 인한 간접부가시간을 절약한다. DOM에 기초한 XML조종방법의 우월성은 6장에서 론의된다.

getXmlString()메쏘드는 목록 5-37에 보여준것과 같은 JSP페지에서 리용하는데 편리하다. 이와 류사하게 main메쏘드는 XML을 파일로 떨구어 검사하기 위한것이다.

목록 5-37에 보여준 간단한 JSP페지(XMLDisplay.jsp)를 리용하여 결과적인 XML을 열람기에 현시할수 있다. <%@ page %>지령문에서 contentType속성의 값을 "text/xml"로 주었다는데 류의하기 바란다. 이 속성은 열람기가 자료를 XML로 인식하고 그에 따라 현시하는데 필요하다.

목록 5-37. JavaBean을 리용하여 ResultSet를 XML로 현시하는 JSP페지

```
<%@ page language="java" contentType="text/xml"%>
<jsp:useBean id="DetailPageXMLBean"
   class="JavaDB_Bible.ch05.sec05.DetailPageXMLBean"
   scope="session"/>
<jsp:setProperty name="Detai1PageXMLBean" property="*"/>
<%=DetailPageXMLBean.getXmlString()%>
```

결과적인 XML을 목록 5-38에 보여준다. 여기에서 보여준 구조가 비록 겹쳐진 요소가 없이 아주 간단하지만 이 실례에서 보여준 모든것들이 보다 복잡한 XML문서들에 동등하게 적용된다.

목록 5-38. XML로 형식화된 ResultSet

```
<?xml version="1.0" encoding="big5"?>
<VehicleData>
   <VehicleID>1001</VehicleID>
   <Body>Convertible</Body>
   <Make>HongKong</Make>
```

```
    <Model>Civic</Model>
    <Engine>Cyllinder</Engine>
    <Color>Red</Color>
    <Transmission>Automatic</Transmission>
    <Price>1000</Price>
    <Year>1995</Year>
    <Options>AM/FM Radio, CD Changer, Moon roof</Options>
</VehicleData>
```

XSL양식일람을 리용한 XML의 변환

양식일람(stylesheet)은 문서를 변환하는데 리용되는 XSL지령들을 포함하고있는 강력한 XML문서이다. XSL양식일람은 XML문서의 뿌리마디를 형성하는 xsl:stylesheet선언으로 시작한다. 양식일람선언은 이름공간과 XML의 판본번호로 이루어진다. 이름공간에서는 아래에 보여주는것처럼 양식일람 태그앞붙이와 태그정의들의 URL을 선언한다.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
</xsl:stylesheet>
```

이름공간의 앞붙이 "xsl:"은 XSL처리문들을 확인하기 위하여 XSL문서의 본체부분에서 리용된다. "xsl:"이라는 앞붙이가 붙지 않은 태그들은 처리되지 않고 간단히 출력되며 따라서 XSL 양식일람에 HTML태그들을 포함시킬수 있다. 그러면 그것들은 변경되지 않고 그대로 출력흐름에 보내여진다.

XSLT는 규칙에 기초한 선언형언어이기때문에 Java와 같은 일반적인 프로그람작성언어들과 다르다. XSL규칙들은 XML문서들이 어떻게 처리되여야 하는가를 규정한 형타들을 정의한다.

XSL형타들은 match연산자를 리용한 처리에서 XML요소들을 선택하는데 리용된다. xsl:template태그의 전형적인 리용실례를 아래에 보여주었다.

```
<xsl:template match="VehicleData">

</xsl:template>
```

match연산자의 인수는 XPath표현식을 리용하여 정의된다. XPath는 파일경로와 꼭 같은 방법으로 자식마디에 대한 경로를 간단히 정의해준다. 실례로 처리중에 있는 전체문

서를 선택하려면 match="/"라고 써서 뿌리마디를 지적할수 있다. 우리의 경우 "VehicleData"자료로 문서요소태그를 맞출수 있다.

경고: "/"로 정의된 뿌리마디맞추기와 "VehicleData"로 정의된 문서마디맞추기에서 XPath에 대한 자식마디는 서로 다르다. 실례로 `<xsl:template match="/">`를 리용하여 뿌리마디를 지적하면 목록 5-37의 뿌리마디에서부터 ID마디까지의 XPath는 "MemberInfo/ID"이다. 한편 문서마디를 맞추기 위하여 `<xsl:template match="MemberInfo">`를 리용하면 XPath는 간단히 "ID"로 된다.

아래의 실례에서 리용된 다른 XSL요소는 xsl:value-of표현식이다. 이 표현식은 선택속성에서 정의된 XPath표현식을 리용하여 선택한 마디의 값을 돌려준다. 실례로 자동차의 색갈을 얻자면 다음의 표현식을 리용한다.

```
<xsl:value-of select="Color"/>
```

이렇게 하면 목록 5-37의 XML문서의 대응하는 마디로부터 "Red"라는 값이 귀환된다.

```
<Color>Red</Color/>
```

XSLT는 XML문서들로부터 값을 불러들이는것과 함께 XML자료를 리용하여 계산을 진행할수도 있으며 문자렬들을 창조 및 조종할수도 있다.

화상의 URL을 만들기 위하여 문자렬을 조종하는 실례를 간단히 보기로 하자.

```
<xsl:variable name="imageUrl"
select=
"string('http://127.0.0.1:8080/MySample/servlet/JavaDB_Bible.ch05.sec
04.LobServlet?id=')"/>
<xsl:variable name="id" select="ID"/>
<img>
   <xsl:attribute name="src">
      <xsl:value-of select="concat($imageUrl, $id)"/>
   </xsl:attribute>
</img>
```

이 코드에서는 문자렬변수를 정의하는 방법과 XML요소들의 값을 그 문자렬과 결합하여 URL을 만드는 방법을 보여주고있다. 그다음 이 URL은 HTML의 `img`태그에서 `src`속성의 값으로 설정된다. 완성된 양식일람을 목록 5-39에 보여주었다. 양식일람이 필요에 따라 XSL과 HTML태그들을 어떻게 자유롭게 결합하는가에 주의하기 바란다.

목록 5-39. XSL stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

<xsl:output method="html"/>
<xsl:preserve-space elements="*"/>

<xsl:template match="VehicleData">
<HTML>
<HEAD>
  <TITLE>Detail Page</TITLE>
  <BASEFONT FACE="Arial"/>
</HEAD>
<BODY>
  <p/>
  <TABLE BORDER="1" WIDTH="480" CELLPADDING="4">
    <TR>
    <TD ALIGN="CENTER" VALIGN="TOP">
      <xsl:variable name="imageUrl"
      select = "string('http ://127.0.0.1:8080/MySample/
                servlet/JavaDB_Bible.ch05.sec05.LobServlet?id=')"/>
      <xsl:variable name="id" select="ID"/>
      <img>
        <xsl:attribute name="src">
          <xsl:value-of select="concat($imageUrl, $id)"/>
        </xsl:attribute>
      </img>
    </TD>
    <TD>
      <xsl:value-of select="Color"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="Year"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="Make"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="Model"/>.
      <p/>
      <xsl:value-of select="Engine"/>,
      <xsl:value-of select="Transmission"/>
      Transmission.
      <p/>
```

```
      <xsl:value-of select="OPTI0NS"/>
      <p/>
      $<xsl:value-of select="Price"/>.
      <p/>
      Vehicle is located in Zip code:
      <xsl: value-of seleot="Zip"/>
    </TD>
    </TR>
    <TR>
    <TD COLSPAN="2">
      <FORM method="post" action="/jsp/ProcessMessageForm.jsp"
        target="_self" id="form1" name="form1">
      <INPUT type="hidden">
      <xsl:variable name="id" select="ID"/>
      <xsl:attribute name="memberld">
        <xsl:value-of select="$id"/>
      </xsl:attribute>
      </INPUT>
      <TABLE BORDER="1">
      <TR>
        <TD>
          <FONT COLOR="blue">
          <EM>Contact the seller:</EM>
          </FONT>
        </TD>
      </TR>
      <TR>
        <TD>
          For more information, or to arrange
          to sees the vehicle, send a message to the seller :
        </TD>
      </TR>
      <TR>
        <TD ALIGN= "CENTER">
          <textarea name="Message" cols = "48" rows="4"/>
        </TD>
      </TR>
      <TR>
        <TD AL1GN="CENTER">
          <input type="submit" value-="Click here to send"/>
        </TD>
      </TR>
```

```
      </TABLE>
      </FORM>
    </TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

JSP페지에서 XSL변환 적용

봉사기측에서 XSL양식일람을 적용하려면 JSP페지를 작성해야 한다. 목록 5-40은 목록 5-36의 JavaBean에 의하여 만들어진 XML문서를 변환하기 위하여 다른 JavaBean의 리용을 보여주고있다. JSP페지에서 리용하는 속성은 회원ID뿐이다.

목록 5-40. JSP페지에서 XSL양식일람을 적용

```
<%@ page language="java"%>
<jsp:useBean id="DetailPageXMLBean"
    class = "JavaDB_Bible.ch05.sec05.DetailPageXMLBean"/>
<jsp:useBean id="DetailPageTransformBean"
    class = "JavaDB_Bible.ch05.sec05.DetailPageTransformBean"/>
<jsp:setProperty name="DetailPageXMLBean" property="*"/>
<%
    DetailPageTransformBean.setXslFileName("DetailPage.xsl");
%>
<%=new String(DetailPageTransformBean.applyTransform(
    DetailPageXMLBean.getVehicleData()))
%>
```

XSL변환을 적용하는데 필요한 JavaBean을 목록 5-41에 보여주었다.

경고: 배비과정에 제기될수 있는 문제는 개발자가 양식일람의 경로를 완전히 규정하지 않으면 그것이 Tomcat의 /bin등록부로 된다.

목록 5-41. XSL변환 bean

```
package JavaDB_Bible.ch05.sec05;

import java.io.*;
import org.xml.sax.SAXException;
import org.apache.xalan.xslt.XSLTProcessorFactory;
```

```
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;
import org.apache.xalan.xslt.XSLTProcessor;

public class DetailPageTransformBean{
   private String xslFileName = null;
   private byte[] xmlSource = null;
   private ByteArrayInputStream xmlInputStream = null;

   public DetailPageTransformBean(){
   }

   public void setXmlSource(byte[] xmlSource){
      this.xmlSource=xmlSource;
      xmlInputStream = new ByteArrayInputStream(xmlSource);
   }

   public void setXslFileName(String xslFileName){
      this.xs1Fi1eName=xslFi1eName;
      File f = new File(xslFileName);
      if(!f.exists())
        System.out.println("Cannot find file: "+xslFileName);
   }

   public byte[] applyTransform(byte[] xmlSource){
      setXmlSource(xmlSource);
      return applyTransform();
   }

   public byte[] applyTransform(){
      ByteArrayOutputStream outStream = new ByteArrayOutputStream();
      try{
         XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
         processor.process(new XSLTInputSource(xmllnputstream),
            new XSLTInputSource(xslFileName),
            new XSLTResultTarget(outStream));
      }catch(Exception e){
         System.err.println(e);
      }
      return outStream.toByteArray();
   }

   public static void main(String args[]){
      File f = new File("Detail.html");
      int id = 1000;
```

```
        DetailPageXMLBean xmlBean = new DetailPageXMLBean();
        DetailPageTransformBean transformBean =
              new DetailPageTransformBean();
        xmlBean.setId(id);
        transformBean.setXslFileName("DetailPage.xsl");

        try {
           FileOutputStream fos = new FileOutputStream(f);
           fos.write(
              transformBean.applyTransform(xmlBean.getVehicleData()));
        }catch(Exception e){
           e.printStackTrace();
        }
    }
}
```

우에서 지적한 모든것을 다 정확히 배비했다면 JSP페지를 호출할 때 그림 5-14와 같은 웨브페지를 보게 된다. 이것을 검사하는 가장 간단한 방법은 아래와 같이 단순한 HTML폼을 작성하는것이다.

```
<html>
<head>
   <title>Get Web Page</title>
</head>
<body>
<form method="POST" action="GetDetailPage.jsp" target="_self">
<table>
   <tr>
      <td align="center" colspan="3">
         <input type="text" name="id">
      </td>
   </tr>
   <tr>
      <td align="center" colspan="3">
         <input type="submit" value="Show Web Page">
      </td>
   </tr>
</table>
</form>
</body>
</html>
```

그림 5-14. ResultSet로부터 만든 XML문서에 XSL변환을 적용하여 만든 웨브페지

실지프로그람에서 상세페지를 호출하려면 목록 5-33의 JSP페지를 간단히 변경한다. 즉 사용자가 검색폼에서 자그마한 화상을 클릭하면 상세페지로 이동한다. 작은 화상우에서 마우스를 찰칵하여 상세페지로 이동하려면 아래와 같이 하면 된다.

```
<TR>
   <TD>
   <A HREF="GetDetailPage.jsp?memberId=
      <%=SearchFormBean.getId()%>">
   <img                             src                              =
"http://127.0.0.1:8080/MySample/servlet/JavaDB_Bible.ch05.sec05.LobSe
rvlet?type=blob&id=<%=(i+1)%>&description=Thumbnail">
   </A>
   </TD>
```

우의 실례에서는 <%=SearchFormBean.getId()%>태그를 리용하여 JSP페지에 회원의 ID를 보낸다.

5.5.3. 갱신가능한 ResultSet와 XSL양식일람의 리용

ResultSet는 현시목적외에 자료기지를 갱신하는데 아주 효과적으로 리용할수 있다. 갱신가능한 ResultSet가 바로 그러한 능력을 제공한다.

XML과 XSLT에 기초하여 웨브페지를 만드는 방법은 갱신가능한 ResultSet들을 리용하는데 아주 적합하다. XSL의 우점들중의 하나가 서로 다른 양식일람들을 리용하여 같은 XML로부터 완전히 서로 다른 Web페지를 만들수 있다는것이다. 이것을 례증하기 위하여 목록 5-42에서 보여준 양식일람을 목록 5-38의 본래 XML에 적용해본다.

목록 5-42. 같은 XML로부터 서로 다른 웨브페지의 창조

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:output method="html"/>
<xsl:template match="VehicleData">
<HTML>
<HEAD>
   <TITLE>
   Edit Detail Page
   </TITLE>
</HEAD>
<BASEFONT FACE="Arial"/>
<BODY>
<FORM method="post" action="ProcessVehicleUpdateForm.jsp">
<TABLE BORDER="1" CELLPADD1NG="4">
<TR>
   <TD>Color</TD>
   <TD>
      <INPUT type="text" name="color">
      <xsl:attribute name="value">
         <xsl:value-of select="Color"/>
      </xsl:attribute>
      </INPUT>
   </TD>
</TR>
<TR>
   <TD>Year</TD>
   <TD>
      <INPUT type="text" name="year">
      <xsl:attribute name="value">
         <xsl:value-of select="Year"/>
      </xsl:attribute>
```

```
        </INPUT>
    </TD>
</TR>
<TR>
    <TD>Make</TD>
    <TD>
        <INPUT type="text" name="make">
        <xsl:attribute name="value">
            <xsl:value-of select="Make"/>
        </xsl:attribute>
        </INPUT>
    </TD>
</TR>
<TR>
    <TD>Model</TD>
    <TD>
        <INPUT type="text" name="model">
        <xsl:attribute name="value">
            <xsl:value-of select="Model"/>
        </xsl:attribute>
        </INPUT>
    </TD>
</TR>
<TR>
    <TD COLSPAN="2">
        <INPUT type="submit" value="CLICK HERE TO SUBMIT CHANGES"/>
    </TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

이 양식일람이 생성하는 폼을 그림 5-15에 보여주었다. XML파일로부터 자료를 얻기 위하여 같은 <xsl:value-of>태그를 쓰고있다. 이때 이 태그는 HTML폼안에서 <xsl:attribute>태그에 둘러싸여있으며 따라서 양식일람은 자동차에 대한 자료를 즉시 현시하는것이 아니라 폼을 미리 적재하는데 리용한다.

경고: Bean의 속성은 대소문자를 구별한다. 속성이름에는 소문자를 리용한다.

Color	
Year	
Make	
Model	
CLICK HERE TO SUBMIT CHANGES	

그림 5-15. 목록 5-42의 양식일람을 리용하여 목록 5-38의 XML로부터 생성한 폼

이 실례에서는 기초를 이루는 XML로부터 아주 적은수의 요소들을 현시할뿐이다. 명백히 편집을 위해 전체문서를 현시하는 하나의 큰 폼을 만들수도 있고 그림 15-4의 실례에서와 같이 그것들을 련속적으로 처리하기 위하여 일련의 보다 작은 폼들을 만들수도 있다.

갱신을 다루기 위하여 요구된 JSP페지를 목록 5-43에 보여주었다. 폼으로부터 얻어낸 속성들을 단순히 UpdateXMLBean에 넘겨주고 bean의 updateVehicleData()메쏘드를 호출한다. 완성되면 사용자는 변경된 결과를 보기 위하여 Web의 상세페지로 가게 된다.

목록 5-43. 자료기지갱신폼을 처리하는 JSP(UpdateXML.jsp)

```
<%@ page language="java" contentType="text/html"%>
<jsp:useBean id="UpdateXMLBean"
   class="JavaDB_Bible.ch05.sec05.UpdateXMLBean"
   scope="session"/>
<jsp:setProperty name="UpdateXMLBean" property="*"/>
<%UpdateXMLBean.updateVehicleData();%>
<%
   String id = UpdateXMLBean.getVehicleId();
   String nextPage = "GetDetailPage.jsp?DetailId=" + id;
%>
<jsp:forward page="<%=nextPage%>"/>
```

자동차자료를 갱신하는 JavaBean을 목록 5-44에 보여주었다. 이 JavaBean은 례외와 함께 목록 5-36의 코드와 비슷하며 갱신가능한 ResultSet를 생성하고 갱신을 수행하는 메쏘드를 포함한다.

목록 5-44. 갱신가능한 ResultSet bean

```
package JavaDB_Bible.ch05.sec05;

import java.io.*;
import java.sql.*;
import javax.sql.*;

public class UpdateXMLBean {
   protected static String dbUserName = "sa";
   protected static String dbPassword = "dba";
   protected String xmlHeader = "<?xml version=\"1.0\"?>";
   protected String vehicleID;
   protected String make;
   protected String model;
   protected String color;
   protected String year;

   protected Connection con;
   protected Statement stmt;
   protected ResultSet rs;
   protected ResultSetMetaData md;

   public UpdateXMLBean() {
   }

   public void setVehicleId(String vehicleID) {
      this.vehicleID = vehicleID;
   }

   public String getVehicleId() {
      return vehicleID;
   }

   public void setYear(String year) {
      this.year = year;
   }

   public String getYear() {
      return year;
   }

   public void setMake(String make) {
```

```
        this.make = make;
    }

    public String getMake() {
        return make;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public String getModel() {
        return model;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public String getVehicleXmlString() {
        String xml = new String(getVrhicleData());
        return xml.trim();
    }

    public String updateVehicleData() {
        String status = "Update successful";
        System.out.println("ResultSet = " + rs);
        try {
            if (rs.getConcurrency() == ResultSet.CONCUR_UPDATABLE) {
                System.out.println("UPDATABLE");
                int nColumns = md.getColumnCount();
                rs.updateString("color", color);
                rs.updateString("make", make);
                rs.updateRow();
            } else {
                System.out.println("READ_ONLY");
                status = "Update failed";
            }
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
    return status;
}

public byte[] getVrhicleData() {
    String rootTag = "VehicleInfo";
    String SQLQuery = "SELECT v.*, o.list AS Options " +
                "FROM Vehicles v, Options o " +
                "WHERE o.VehicleID = v.VehicleID AND " +
                "v.VehicleID = '" + vehicleID + "';";
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    try {
        Class.forName("com.inet.pool.PoolDriver");
        com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
        tds.setServerName("DOLPHIN");
        tds.setDatabaseName("MEMBERS");
        tds.setUser(dbUserName);
        tds.setPassword(dbPassword);

        DataSource ds = tds;
        Connection con = ds.getConnection(dbUserName, dbPassword);

        stmt = con.createStatement(
                ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
        rs = stmt.executeQuery(SQLQuery);
        md = rs.getMetaData();
        if (rs.getConcurrency() == ResultSet.CONCUR_UPDATABLE) {
            System.out.println("UPDATABLE");
        } else {
            System.out.println("READ_ONLY");
        }
        os.write(xmlHeader.getBytes());
        os.write(("<" + rootTag + ">").getBytes());

        String xml = "";
        int columns = md.getColumnCount();
        rs.next();
        for (int i = 1; i <= columns; i++) {
            if (md.getColumnType(i) == Types.VARCHAR) {
```

```
                xml = "<" + md.getColumnLabel(i) + ">" +
                    rs.getString(i) + "</" + md.getColumnLabel(i) + ">";
                os.write(xml.getBytes());
            }
        }
        os.write(("</" + rootTag + ">").getBytes());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return os.toByteArray();
  }
}
```

HTML폼의 속성들에 설정자메쏘드들이 보인다는데 주목할 필요가 있다. 만일 매개폼속성에 대한 설정자메쏘드를 포함하지 않는다면 Tomcat는 "method not found"라는 오유통보문을 내보낸다. 또한 갱신이 정확히 진행되였다고 가정하기전에 `ResultSet.getConcurrency()`메쏘드를 리용하여 `ResultSet`가 실지로 갱신가능한것인가를 검사하는것이 중요하다.

제6절. JDBC와 전자우편

Internet에서 사용되던 이전의 전자우편은 주로 간단한 Java프로그람으로 조종할수 있는 본문통보문으로 구성되였다. 그러나 전자우편의 인기가 급속히 증가함에 따라 그의 능력이 확장되였으며 오늘날 대부분의 전자우편들은 본문과 HTML의 두가지 형식으로 전송되며 각이한 자료형들을 포함할수 있게 되였다.

JavaMail API는 Java를 리용하여 우와 같은 보다 복잡한 전자우편통보문들을 조종하기 위하여 개발되였다. 이 절에서는 JavaMail API에 대하여 개괄하고 전자우편을 보내고 받는데 JDBC와 JavaMail을 리용하는 방법을 설명한다.

5.6.1. 전자우편규약

전자우편의 골간망은 전자우편들을 보관하고 이동시키는 역할을 하는 호상접속된 단순우편전송규약(Simple Mail Transfer Protocol: SMPT)봉사기들의 망이다. 전자우편을 전송하려면 자기의 국부 SMTP봉사기에 접속하고 SMTP를 리용하여 전자우편을 발송한다. 그러면 전자우편은 수신자의 봉사기에로 전송되여 수신자의 전자우편 등록부에

들어간다. 수신자는 보통 우편국규약(Post Office Protocol: POP)를 리용하여 자기에게 온 전자우편들을 검색한다.

전자우편통보문들이 점점 복잡해지면서 전자우편통보문들에 포함된 여러가지 자료형들을 관리해야 할 요구가 제기되였다. 이로부터 MIME이 개발되였다.

MIME는 전자우편통보문, 부속물, 기타 등등의 내용을 정의한다. MIME자료형은 통보문의 기본부분에서 자료의 형과 부분형을 규정하는데 리용되는 `Content-Type`머리부마당에서 정의된다. 일반적인 MIME형들은 다음과 같다.

- text 표준본문내용을 표현하는데 리용된다.
- multipart 각이한 형의 여러 본체부분들을 단일한 통보문으로 결합하는데 쓰인다.
- application 프로그람자료나 2진자료를 전송하는데 리용
- image 정지화상자료(그림)를 전송하는데 리용
- audio 음향자료(audio)나 음성자료를 전송하는데 리용
- video 비데오 혹은 동화상자료를 전송하는데 리용

JavaMail API를 리용하는 사용자는 통보문의 머리부에서 MIME형을 얻어내여 통보문을 어떻게 처리할것인가를 결정하는데 리용할수 있다.

SMTP는 보통 인터네트봉사제공자(ISP:Internet Service Provider)가 관리하는 SMTP봉사기에 전자우편을 보내는데 리용된다. 이 SMTP봉사기는 수신자의 SMTP봉사기에 전자우편통보문를 위임하며 수신자의 봉사기는 수신자가 그것을 찾아갈수 있도록 보관한다.

POP의 현재 판본번호가 3이므로 이 규약을 POP3이라고도 부른다. POP3은 매 리용자들마다에 단일한 우편통을 지원하므로 전자우편을 내리적재하는데 가장 널리 리용되는 방법이다.

주의: POP3은 전자우편의 기본적인 저장과 내리적재만을 지원한다. 새로 받은 전자우편의 추적과 같은 기능은 Endora와 같은 의뢰기들에서 조종된다.

5.6.2. JavaMail API의 리용

전자우편을 조종하는 가장 좋은 방법은 JavaMail API를 리용하는것이다. JavaMail API는 통보문을 보내고 받는데서 규약에 의존하지 않는 방법들을 제공하도록 설계되였다.

JavaMail을 리용하여 전자우편을 전송하는데서 첫번째 단계는 JavaMail `Session`을 얻는것이다. 그 `Session`안에서 새로운 `Message`객체를 하나 창조하고 그 속성들을 설정

한 다음 그것을 전송한다. 아래에 이러한 과제들을 수행하는데 필요한 핵심적인 JavaMail API클라스들을 렬거한다.

- Session　기본 우편대화접속을 정의한다.
- Message　대부분의 경우에 javax.mail.internet.MimeMessage를 리용한다.
- Address　표준적으로 javax.mail.internet.InternetAddress를 리용한다.
- Transport　통보문을 전송하는데서 포함된 규약에 따르는 과제들을 수행한다.
- Store　전자우편통보문들을 받기 위해서는 먼저 Mail Store에 접속해야 한다.
- Folder　Mail Store는 내리적재되고 읽을수 있는 통보문들의 등록부를 포함하고있다.

핵심 JavaMail API의 session, message, address, transport클라스들에 대해서는 첫번째 전자우편전송실례에서 구체적으로 설명한다. 나머지 클라스들은 전자우편통보문들을 받을 때 리용되며 두번째 전자우편받기실례에서 설명한다.

Session객체는 기본적인 우편대화접속을 정의한다. 이 객체는 우편봉사기, 사용자이름, 통과암호와 같은 응용프로그람준위의 정보를 유지하는 java.util.Properties객체를 리용한다. 대부분의 경우 다중사용자우편통과 작업한다해도 공유된 대화접속을 리용할수 있다.

Message객체는 전자우편통보문을 나타낸다. Message객체의 속성들은 우편물의 제목, 내용, 송신자와 수신자의 주소들을 포함하고있다. MimeMessage는 여러가지 MIME형들과 머리부들을 인식하는 전자우편통보문이다.

전자우편의 주소는 Address객체를 리용하여 실현된다. Address객체는 표준적으로 javax.mail.internet.InternetAddress클라스를 리용하여 창조한다. Address객체는 전자우편주소만을 설정하게 하거나 송신자와 수신자의 전자우편주소와 이름을 설정하게 하는 구축자들을 가지고있다.

주의: JavaMail API는 Address객체의 내용을 검사하지 않는다. 따라서 우편봉사기가 방해하지 않는한 통보문전송은 중지되지 않는다.

Transport객체는 통보문전송을 위한 규약특정의 언어(보통 SMTP)를 다룬다. 개발자는 정적메쏘드인 send()를 호출하여 클라스의 기정판본을 리용할수도 있고 대화접속으로부터 특정한 구체례를 얻을수도 있다. 아래에 한가지 실례를 보여준다.

```
Transport transport = session.getTransport("smtp");
transport.connect(host, username, password);
transport.sendMessage(message, message.getAllRecipients());
transport.close();
```

주의: 기초적인 send()물림새는 매 메쏘드호출에 대하여 봉사기에 대한 별도의 접속을 만든다. 통보문들을 여러개 전송해야 할 때에는 Transport의 특정한 구체례를 얻는것이 더 좋다. 이것은 통보문들사이에 능동인 우편봉사기로 접속을 유지한다.

5.6.3. JDBC와 JavaMail에 의한 전자우편 보내기

자료기지구동형 전자우편의 가장 일반적인 리용의 하나는 자기의 통과암호를 잊어먹은 사용자들을 도와주는것이다. 이것은 간단한 프로그람이며 JSP페지와 JavaBean으로 쉽게 처리할수 있다.

자기의 통과암호를 잃어버린 회원에게 전자우편을 보내려면 그 회원의 전자우편주소를 검색하기 위하여 Contact_Info표에 질문해야 한다. 또한 Login표에서 그의 통과암호도 얻어야 한다.

```
SELECT l.password, c.email
   FROM LOGIN l, CONTACT_INFO c
   WHERE l.username = 'KimSongChol' AND
   l.MemberID = c.MemberID;
```

이 실례에서 목적하는것은 그 회원의 통과암호와 짤막한 해설문을 내용으로 하는 간단한 통보문이 전부이다. 자료기지의 Contact_Info표에서 얻어낸 전자우편주소는 Message객체의 recipient속성에 삽입된다.

이 실례는 2절에서 개발한 회원자료기지를 리용하여 JSP로 개발된다. 가입등록 JSP페지를 설정하여 가입등록검사에서 실패한 사용자를 이제부터 개발하게 되는 SendMailBean을 리용하는 JSP페지로 안내한다.

전자우편전송을 위한 JDBC와 JavaMail의 리용

SendMailBean은 목록 5-10에서 본 LoginBean에 JavaMail성분을 추가한것과 비슷하다. 실례에서는 DataSource객체를 리용하여 자료기지에 접속하며 그 회원의 통과암호와 전자우편주소를 검색한다. 만일 전자우편주소가 "null"이 아니라면 emailPassword()메쏘드를 호출하여 사용자에게 통과암호를 보낸다.

JavaMail에 기초한 emailPassword()메쏘드의 내부작업은 간단하다. 첫 단계는 체

계의 properties객체를 얻고 전자우편주콤퓨터의 이름을 삽입하는것이다.

```
props.put("mail.smtp.host", host);
```

다음 단계는 전자우편이 전송되는 전후상황을 제공하는 Session객체를 얻는것이다.

```
Session session = Session.getDefaultInstance(props, null);
```

일단 Session객체를 얻으면 그것을 리용하여 아래와 같이 MimeMessage객체를 생성한다.

```
MimeMessage message = new MimeMessage(session);
```

이제 남은것은 Message객체의 속성들을 설정하고 그것을 보내는것이다. Message객체를 받는 형식에는 Message.Recipient.TO외에 다음의 형식들이 있다.

- Message.Recipient.CC
- Message.Recipient.BCC

```
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
  new InternetAddress(email));
message.setSubject("Password Reminder");
message.setText("Hi " + memberName + ", Your password is: " + password);
```

끝으로 Message객체를 인수로 하여 Transport.send()메쏘드를 호출한다.

```
transport.send(message);
```

목록 5-45에 SendMailBean을 구체적으로 보여주었다.

목록 5-45. JavaMail API와 JDBC를 리용한 전자우편의 전송

```
package JavaDB_Bible.ch05.sec06;

import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SendMailBean {
    private static String dbUserName = "sa";
    private static String dbPassword = "dba";
```

```
private Connection con = null;
protected String username;

public SendMailBean() {
}

public void setUsername(String username) {
    this.username = username;
}

public String getUsername() {
    return username;
}

public String getPasswordAndEmailAddress() {
    String password = null;
    String email = null;

    try {
        Class.forName("com.inet.pool.PoolDriver");
        com.inet.tds.TdsDataSource tds =
                new com.inet.tds.TdsDataSource();
        tds.setServerName("DOLPHIN");
        tds.setDatabaseName("MEMBERS");
        tds.setUser(dbUserName);
        tds.setPassword(dbPassword);

        DataSource ds = tds;
        Connection con = ds.getConnection(dbUserName, dbPassword);
        String SQLQuery = "SELECT l.Password, c.Email " +
                     "FROM LOGIN l, CONTACT_INFO c " +
                     "WHERE l.MemberID = c.MemberID " +
                     "AND l.Username = '" + username + "';";
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(SQLQuery);
        while (rs.next()) {
            password = rs.getString("Password");
            email = rs.getString("Email");
        }
        con.close();
    } catch (ClassNotFoundException e1) {
        System.err.println(e1.getMessage());
    } catch (SQLException e2) {
        System.err.println(e2.getMessage());
    }
```

```
        if (email == null) {
            return "Bad Email";
        } else {
            emailPassword(email, username, password);
            return "OK";
        }
    }

    public void emailPassword(String email,
                              String memberName, String password) {
        String host = "mail";
        String from = "dolphin@sec.com";

        Properties props = System.getProperties();

        props.put("mail.smtp.host", host);

        // session얻기
        Session session = Session.getDefaultInstance(props, null);

        // message정의
        MimeMessage message = new MimeMessage(session);

        try {
            // 송수신자주소 설정
            message.setFrom(new InternetAddress(from));
            message.addRecipient(Message.RecipientType.TO,
                                 new InternetAddress(email));

            // 제목설정
            message.setSubject("Password Reminder");

            // message내용 설정
            message.setText("Hi " + memberName +
               ",\nYour password is: " + password + "\nregards - " + from);

            // 전송
            Transport.send(message);
        } catch (AddressException ae) {
        } catch (MessagingException me) {
        }
    }
}
```

Login표가 UserName으로 색인되고 Contact_Info표가 MemberID로 색인되여있기 때문에 이 실례에서 SQL질문은 매우 효률적이다.

SendMailBean을 리용하기 위한 JSP페지

SendMailBean을 리용할 때에는 JSP페지에서 UserName인수를 설정하고 SendMailBean.getPasswordAndEmail()을 호출하여 자료기지에 질문하고 전자우편을 보낸다. SendMailBean을 리용하는데 필요한 JSP페지를 목록 5-46에 보여주었다.

목록 5-46. SendMailBean을 리용하기 위한 JSP페지(SendMail.jsp)

```
<html>
<head>
   <title>Email Password</title>
</head>
<body>
   <%@ page language="java"%>
   <jsp:useBean id="SendMailBean" scope="application"
      class="JavaDB_Bible.ch05.sec06.SendMailBean"/>
   <jsp:setProperty name="SendMailBean" property="*"/>
   <%
      String userName = request.getParameter("username");
      String emailStatus = SendMailBean.getPasswordAndEmailAddress();
      if(emailStatus.equals("OK")){
   %>
   Hi <%=userName%>,<br>
   Your password is being emailed to the address we have on file.
   <%
      }else{
   %>
   Sorry, <%=userName%>, <br>
   Your email address is not on file.
   <%
      }
   %>
</body>
</html>
```

JSP페지와 함께 리용하는 JavaBean을 배비하기 위해서는 bean클라스파일들을 적당한 등록부에 놓아야 한다. 단순한 Tomcat설치에 대한 보통 경로는 다음과 같다.

```
TOMACT/WEBAPPS/ROOT/WEB-INF/CLASSES
```

현재 우리의 경우에는 Tomcat\webapps\MySample\WEB-INF\classes이다.

써블레트를 배비하려면 요구되는 jar파일들을 적당한 등록부로 옮기고 Tomact/conf 등록부에 있는 tomcat.properties파일에서 Tomcat의 클라스경로를 변경해야한다. 이 실례에서는 jar파일이 /lib등록부에 보관되여있으므로 tomcat.properties파일에 다음의 행들을 추가하면 Tomcat의 클라스경로가 변경된다.

```
wrapper,classpath=lib/jdbc2_0-stdext.jar
wrapper.classpath=lib/activation.jar
wrapper.classpath=lib/mail.jar
wrapper.claaspath=lib/Opta2000.jar
```

또는 체계의 환경변수설정에서 우의 jar파일들을 CLASSPATH에 추가해준다.

5.6.4. JavaMail API를 리용한 전자우편 받기

JavaMail API로 전자우편을 받는것은 전자우편을 보내는것보다 약간 복잡하다. 전자우편을 받을 때에는 보낼 때 쓰인 JavaMail객체들외에 Store객체와 Folder객체의 리용이 더 포함된다.

전자우편을 받을 때 일어나는 사건들은 전자우편을 보낼 때와 류사하다.

1. 기정의 전자우편대화접속을 얻는다.
2. POP3 통보문저장객체를 얻는다.
3. 봉사기이름과 우편사용자이름, 통과암호를 리용하여 저장고(store)에 접속한다.
4. 기정 폴더를 얻는다.
5. INBOX를 얻는다.
6. INBOX를 열고 통보문들을 읽는다.

처리는 통보문을 보낼 때와 거의 같은 방법으로 시작한다. 그러나 대화접속을 얻은 다음 Transport대신 Store에 접속한다. 간단히 실례를 들어보자.

```
Store store = session.getStore("pop3");
store.connect(host, username, password);
```

Store에 접속한 다음에는 폴더를 얻고 그것을 연다. POP3을 사용할 때 리용가능한 유일한 폴더는 INBOX이다. 이 폴더를 열면 그로부터 통보문들을 읽을수 있다. 아래에 그 방법을 보여주었다.

```
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message message[] = folder.getMessages();
```

folder.getMessages()메쏘드는 느린 자료검색방법(lazy data retrival)을 리용한다. 다시 말하여 통보문의 내용은 특별히 요구될 때에만 내리적재된다. 통보문의 내용은 getContent()메쏘드로 얻을수 있으며 writeTo()메쏘드로 그것을 흐름에 쓸수 있다. getContent()메쏘드로 얻은 통보문의 내용을 writeTo()메쏘드로 출력할 때에는 머리부들이 포함된다.

```
System.out.println(((MimeMessage)message).getContent());
```

INBOX폴더가 읽기전용으로 열린다는데 주의해야 한다. 쓰기접근은 통보문에 받았다는 표식을 하거나 그것들을 봉사기에서 지울 때 리용된다. 목록 5-47에서는 JavaMail API를 리용하여 전자우편통보문들을 받는 방법을 보여준다.

목록 5-47. JavaMail로 전자우편을 읽고 그것을 자료기지에 보관

```
package JavaDB_Bible.ch05.sec06;

import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;
import java.sql.*;
import javax.sql.*;

public class JavaMailReceiver {
   static String server = "mail.home.com";
   static String username = "Girl";
   static String password = "java";

   static MailSaver db = new MailSaver();

   public static void main(String args[]) {
      try {
         receive(server, username, password);
      } catch (Exception e) {
         System.err.println(e);
      }
      System.exit(0);
   }
```

```
public static void receive(String server,
                 String username, String password) {
   Store store = null;
   Folder folder = null;

   try {
      // 기정session 얻기
      Properties props = System.getProperties();
      Session session = Session.getDefaultInstance(props, null);

      // POP3 통보문저장객체 store얻고 그에 접속
      store = session.getStore("pop3");
      store.connect(server, username, password);

      // 기정 folder얻기
      folder = store.getDefaultFolder();
      if (folder == null)throw new Exception("No default folder");

      // INBOX얻기
      folder = folder.getFolder("INBOX");
      if (folder == null)throw new Exception("No POP3 INBOX");

      // 폴더를 읽기전용으로 열기
      folder.open(Folder.READ_ONLY);

      // 통보문처리
      Message[] msgs = folder.getMessages();
      int msgNum = msgs.length;
      while (processMessage(msgs[--msgNum]));
   } catch (Exception e) {
      e.printStackTrace();
   } finally {
      try {
         if (folder != null) folder.close(false);
         if (store != null) store.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}

public static boolean processMessage(Message message) {
   Calendar today = Calendar.getInstance();

   try {
```

```
        // 머리부정보 얻기
        String subject = message.getSubject();
        String dateString = "unknown date";
        String to = ((InternetAddress)
               message.getAllRecipients()[0]).getPersonal();
        String toEmail = ((InternetAddress)
               message.getAllRecipients()[0]).getAddress();
        String from = ((InternetAddress)
               message.getFrom()[0]).getPersonal();
        String email = ((InternetAddress)
               message.getFrom()[0]).getAddress();

        if (to == null) to = toEmail;
        if (from == null) from = email;

        java.util.Date date = message.getSentDate();
        Calendar mDate = Calendar.getInstance();

        if (date != null) {
           dateString = date.toString();
           mDate.setTime(date);
           if (mDate.get(Calendar.DAY_OF_MONTH) <
              today.get(Calendar.DAY_OF_MONTH) - 3) return false;
        }

        System.out.println("DATE: " + dateString);
        System.out.println("TO: " + to + " <" + toEmail + ">");
        System.out.println("FROM: " + from + " <" + email + ">");
        System.out.println("SUBJECT: " + subject);

        // 통보문 얻기
        Part messagePart = message;
        Object content = messagePart.getContent();

        if (content instanceof Multipart) {
           for (int i=0;i<((Multipart) content).getCount();i++) {
              messagePart = ((Multipart) content).getBodyPart(i);
              String contentType = messagePart.getContentType();

              if (contentType.startsWith("text/plain") ||
                 contentType.startsWith("text/html")) {
                 String msg = readMsg(messagePart);
                 db.saveEmail(dateString, from, email, subject,
                          contentType, msg);
              }
           }
```

```
            } else {
               String contentType = messagePart.getContentType();
               if (contentType.startsWith("text/plain") ||
                  contentType.startsWith("text/html")) {
                  String msg = readMsg(messagePart);

db.saveEmail(dateString,from,email,subject,contentType,msg);
               }
            }
         } catch (Exception ex) {
            ex.printStackTrace();
         }
         return true;
      }

      private static String readMsg(Part messagePart) {
         String message = "";
         try {
            String contentType = messagePart.getContentType();
            if (contentType.startsWith("text/plain") ||
               contentType.startsWith("text/html")) {
               InputStream is = messagePart.getInputStream();
               BufferedReader reader = new BufferedReader(new
                     InputStreamReader(is));
               String line = reader.readLine();
               while (line != null) {
                  message = message + line;
                  line = reader.readLine();
               }
            }
         } catch (Exception e) {
            System.err.println(e);
         }
         return message;
      }
}

class MailSaver {
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";

   Connection con = null;
   public MailSaver() {
      try {
         Class.forName("com.inet.pool.PoolDriver");
         com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
```

```
            tds.setServerName("DOLPHIN");
            tds.setDatabaseName("MEMBERS");
            tds.setUser(dbUserName);
            tds.setPassword(dbPassword);

            DataSource ds = tds;
            Connection con = ds.getConnection(dbUserName, dbPassword);
        } catch (Exception e) {
            System.err.println("SQL Exception registering driver");
        }
    }

    public void saveEmail(String date, String sender,
                          String senderEmail, String subject,
                          String mimeType, String msg) {
        String cmd = "INSERT INTO EMAIL " +
                     "(MsgDate,Sender,SenderEmail,Subject,ContentType," +
                     "Message) VALUES (?,?,?,?,?,?)";
        try {
            PreparedStatement pstmt = con.prepareStatement(cmd);
            pstmt.setString(1, date);
            pstmt.setString(2, sender);
            pstmt.setString(3, senderEmail);
            pstmt.setString(4, subject);
            pstmt.setString(5, mimeType);
            pstmt.setString(6, msg);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

목록 5-47의 실례는 간단한 JavaMail프로그람이다. 여기서는 SMTP봉사기에 가입등록하고 getMessages()메쏘드를 리용하여 아래와 같이 봉사기에 있는 통보문들을 모두 얻는다.

```
Message[] msgs = folder.getMessages();
int msgNum = msgs.length;
while(processMessage(msgs[--msgNum]));
```

실제적인 통보문처리는 processMessage()메쏘드가 담당한다. 통보문들은 processMessage()메쏘드가 false를 돌려줄 때까지 도착순서 반대로 처리된다는데 주의해야 한다. processMessage()메쏘드는 조종탁에 현시하기 위한 날자와 제목, 송신자의

정보를 분석한다. 다음 날자를 현재 날자와 비교하여 그 통보문이 3일이전에 온것이라면 false를 돌려준다.

만일 통보문의 MIME내용이 본문이나 HTML이라면 이 통보문은 간단한 E-mail표에 Clob로 보관된다. E-mail표는 다음과 같은 렬들을 포함하고있다.

- 날자(Date)
- 송신자(Sender)
- 송신자의 전자우편주소(SenderEmail)
- 제목(Subject)
- 내용부의 자료형(ContentType)
- (우편물)내용(Content)

단일한 통보문의 개별적인 부분들이 서로 다른 행들에 보관되므로 통보문은 하나이상의 행들에 보관될수 있다는데 주의해야 한다. 표에서 자동적으로 증가되는 통보문ID의 리용은 통보문 부분들을 개별적으로 식별하는데 도움된다.

5장에 대한 요약

이 장에서는 써블레트와 JSP로 업무론리를 실현하는 Apache/Tomcat기초의 웨브봉사기를 중심으로 구축된 3층구성방식에 대하여 학습하였다. 이를 위하여 취급된 내용들은 다음과 같다.

- 자료기지접속을 얻기 위한 DataSource객체의 리용방법
- HttpServlet객체의 리용
- JSP에서 JavaBean의 리용
- 3가지 Statement객체의 리용
- 대형객체의 조종
- 흘리기가능한 ResultSet를 리용한 웨브페지 탐색
- ResultSet로부터 XML문서의 작성
- 서로 다른 웨브페지를 창조하기 위한 XSL양식일람의 적용
- 자료기지레코드를 갱신하기 위한 갱신가능한 ResultSet의 리용
- JavaMail API를 리용한 전자우편프로그람의 개발

제 6 장. 자료기지와 JDBC, XML 의 리용

XML은 프로그람내부에서 또는 웨브상에 있는 프로그람들사이에서 자료를 관리하고 교환하기 위한 규격으로 급속히 발전하고있는 본문형 표식언어이다. 얼핏 보면 XML문서가 많은 측면에서 HTML문서와 비슷하지만 두 언어사이에는 중요한 차이가 있다. 이 차이점들중에서 가장 중요한것은 다음과 같다.

- HTML은 주로 본문과 기타 자료에 형식화정보를 가진 표식을 다는데 리용된다.
- XML은 주로 자료운반이나 프로그람내부에서의 국부적인 사용을 위해 자료를 구조화하는데 쓰인다.

다시말하여 HTML문서는 본질적으로 문서위주(즉 웨브페지와 같이 주로 사람의 직접적인 사용을 목적)로 설계되였다. 이와는 달리 XML문서는 자료위주(즉 기계에서의 사용을 목적)로 설계되였다.

자료위주의 문서들은 기계가 생성하는것이기때문에 문서위주의 자료들보다 규칙적이며 째인 구조를 가지는것이 특징이다. 자료위주문서의 내용은 자료기지와 밀접한 련관속에 있다. 즉 XML문서를 발행하는 경우에는 그 내용을 자료기지로부터 읽어들이고 XML문서가 자료운반에 리용되였을 때에는 그 내용이 자료기지에 저장된다. 어떤 경우에는 자료를 저장하고있는 XML문서 그 자체가 자료기지의 역할을 수행한다.

제1절. XML문서객체모형과 JDBC

이 절에서는 XML에 대한 간단한 소개를 준 다음 SQL질문으로부터 XML문서를 생성하는 방법과 XML문서에서 자료기지로 자료를 이식하는 방법에 대하여 고찰한다.

6.1.1. XML과 HTML의 차이

확장표식언어(eXtensible Markup Language - XML)는 본문에 기초한 표식언어로서 프로그람들사이 그리고 웨브상에서의 자료교환규격으로 급속히 발전하고있다.

XML은 목록 6-1에서 보여주는것처럼 자료를 식별하기 위하여 각괄호(<>)로 둘러싸인 태그들을 리용한다는 점에서 초본문표식언어(HTML)와 류사하다.

목록 6-1. XML의 실례

```
<?xml version="1.0"?>
<CONTACT_INFO>
    <FAMILY_NAME>김</FAMILY_NAME>
    <PERSONAL_NAME>성철</PERSONAL_NAME>
    <STATE>평양시</STATE>
    <CITY>평양</CITY>
    <STREET>창광거리</STREET>
    <PHONE>359-6532</PHONE>
</CONTACT_INFO>
```

자료를 현시하는 방식을 규정하는 HTML태그와 달리 XML의 태그들은 자료를 식별하고 설명하는데 리용된다. 자료를 식별하고 설명하는데 리용되는 태그들은 응용프로그람에 관계되며 따라서 그것이 W3C XML규격이 정의하는 규칙에 따르기만 한다면 쓰고 싶은 태그들을 임의로 쓸수 있다.

XML과 HTML의 기본차이는 XML문서가 언제나 엄밀하게 구성되여야 한다는것이다. 우선 모든 태그는 닫는 태그를 가져야 한다. 실례로 HTML에서 단락태그 <p>와 행바꾸기태그
를 자주 보게 된다. 매 태그가 항상 닫길것을 요구하는 XML에서는 <p></p> 혹은 <p/>와 같은 형식으로 닫기태그를 꼭 리용하여야 한다.

또한 정확치 않은 겹침이 HTML에서는 일없다고해도 XML에서는 허용되지 않는다. 대부분의 열람기들에서는 HTML태그들이 무질서한 순서로 닫겨있어도 별일없이 조종할수 있다. 아래의 실례에는 요소들이 정확히 닫기지 않거나 전혀 닫기지 않은 태그들도 있다. 이 실례가 열람기에서는 잘 현시되지만 XML로는 읽을수 없다.

```
<HTML>
<BODY>
    <CENTER>
    <FONT FACE="Arial">
    Hello World
    </CENTER>
    </FONT>
```

프로그람을 작성하는 관점에서 보면 XML은 문자흐름이나 객체로 취급할수 있다. 문자흐름에 기초한 분석기에서는 XML요소들이 련속적으로 확인되며 사건구동형처리기의 방아쇠로 리용된다. 문서객체모형이 리용될 때에는 전체 문서가 등록부나무구조에서 파일들이 접근되는 방식과 류사하게 다양한 요소와 속성들이 이름과 경로에 의해 접근될수 있는 문서객체로 분석된다.

6.1.2. XML과 문서객체모형

문서객체모형(Document Object Model - DOM)에서는 XML문서를 나무구조로 표현한다. 문서요소는 나무의 꼭대기준위이다. 문서요소는 나무의 가지들을 표현하는 수많은 자식마디들을 가진다. 그림 6-1에서는 나무구조로 표현된 XML문서를 보여준다.

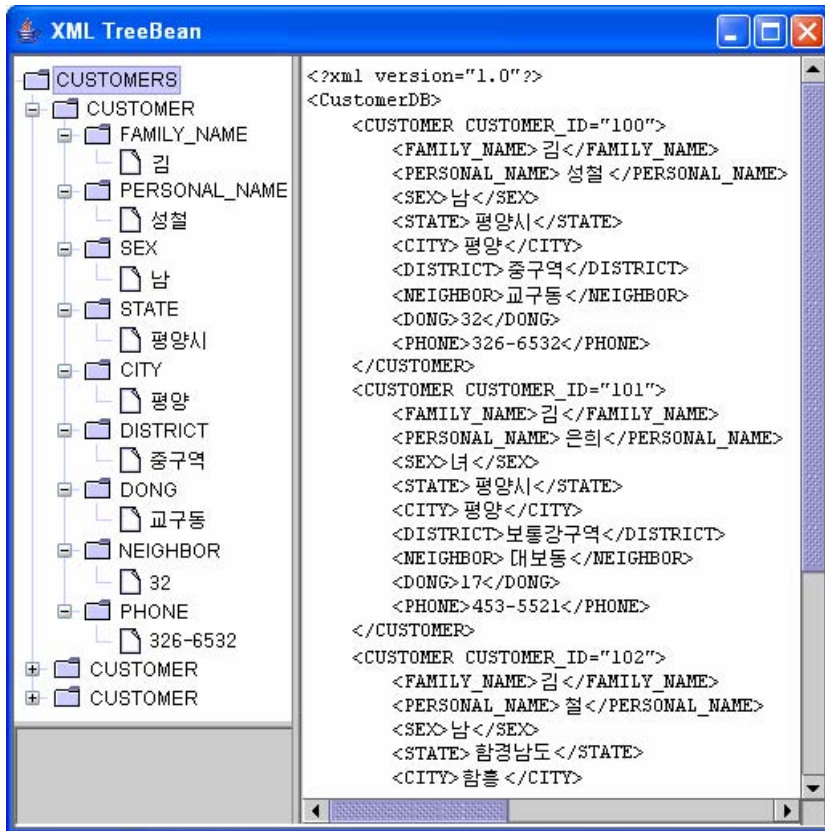



그림 6-1. 나무로 현시한 XML문서

DOM의 가장 기초적인 성분은 마디(Node)대면부이다. XML문서의 DOM표현에서 매 구성성분은 마디이다. 마디대면부는 요소(Element)대면부, 문서(Document)대면부와 같은 다른 대면부들에 의하여 펼쳐질수 있다. 그림 6-1에서 보여준 실례에서 문서요소는 JTree표시에서 강조된 CUSTOMERS요소이다. 오른쪽에 있는 본문표시로부터 CUSTOMERS요소가 속성마디 DBNAME="CONTACTS"를 가지고있다는것을 알수 있다.

CUSTOMERS가 요소마디인것처럼 CUSTOMER, FIRST_NAME도 역시 요소마디이다. JTree는 요소들을 폴더로 표시한다. 요소마디안에는 본문과 함께 문서아이콘으로 표현되는 추가적인 요소마디들이 있다. XML문서를 Java객체들의 나무구조로 표시하면 Java프로그람작성자들이 현재 널리 리용하고있는 많은 API들중 하나를 리용하여 XML문

서들과 그의 내용들을 창조, 접근 및 변경할수 있다. DOM을 론의하기전에 XML문서의 구조를 세부적으로 관찰해보자.

XML문서의 머리부

XML파일은 항상 이 문서가 XML이라는것을 식별하게 하는 선언문으로 시작한다. 가장 작은 머리부는 다음과 같다.

```
<?xml version="1.0"?>
```

선언문에는 XML의 판본번호와 문자부호화정보 그리고 문서류형정의(Document Type Definition: DTD)와 기타 련관된 정보들이 포함될수 있다.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

이 실례의 머리부에는 다음과 같은 정보들이 포함되여있다.

- XML의 판본번호는 1.0이다.
- 문서의 부호화방식은 HTTP기정문자부호화인 ISO-8859-1이다.
- XML문서의 구조와 제약조건을 나타내는 DTD와 같은 문서들에 대한 지원을 전혀 요구하지 않는다.

XML머리부 다음에 오는것은 모두 문서의 내용을 이룬다.

주의: XML의 판본번호속성은 꼭 필요하다. XML분석기는 판본번호속성이 없으면 오유통보를 내보낸다.

태그들과 속성들

목록 6-1의 실례에서 태그들은 접촉자의 성, 이름, 도시, 거리, 전화번호 등 개별적인 요소들은 물론 전체로서 내용을 식별한다. 이 자료요소들은 <CONTACT_INFO>태그안에 그것들을 겹쳐넣는것으로 정의되는 계층구조속에 포함된다. 한 태그안에 다른 태그들을 포함하는 능력은 XML이 계층화된 자료구조를 표현할수 있게 한다.

인쇄된 페지상에서 XML문서의 형식화는 주로 편리상의 문제이다. HTML의 경우와 마찬가지로 공백부분은 그리 중요하게 고려되지 않는다.

XML태그들은 태그의 각괄호안에 태그이름외에 속성들도 포함한다. HTML에서처럼 속성들은 일반적으로 그 요소에 대한 추가적인 정보를 제공하는데 리용된다. 속성들을 가지고있는 태그의 좋은 실례가 아래에 보여주는것처럼 HTML의 <FONT>태그이다. 여기에는 서체의 이름, 크기, 색과 같은 속성들이 포함되여있다.

```
<FONT FACE="Arial" SIZE="3" COLOR="#0000FF">Hello World</FONT>
```

XML에서도 HTML에서와 마찬가지로 속성들이 공백으로 구분되며 "열쇠=값"의 쌍으로 정의된다. 그러나 HTML과 달리 XML에서는 속성들을 공백으로 구분할뿐만아니라 그 속성값을 반드시 인용부호(" ")안에 넣을것을 요구한다. 다시 말하여 우에서 보여준 FONT태그는 XML속성들을 정의하기 위한 요구에 맞지만 아래의 FONT실레는 XML로 리용할수 없다.

```
<FONT FACE=Arial SIZE=3 COLOR=#0000FF>Hello World</FONT>
```

속성이나 태그들을 리용하면 자료구조를 <통지문>과 같이 잘 설계할수 있으므로 개발자는 자기의 목적에 어느 설계가 가장 좋겠는가를 그려내는데 충분한 사색을 할수 있다.

요소와 마디들

DOM은 XML문서를 나무구조로 표시하며 나무구조의 매 마디는 XML문서의 한개 구성요소를 포함하고있다. DOM메쏘드들을 리용하면 마디들의 창조와 삭제, 그것들의 내용변경, 그리고 마디계층을 가로지를수 있다.

DOM은 `org.w3c.dom.Node`대면부안에 각이한 류형의 많은 마디들을 정의한다. 가장 일반적으로 쓰이는것들을 표 6-1에 보여주었다.

표 6-1. org.w3c.dom대면부

org.w3c.dom Node_Type	Application	Example
ATTRIBUTE_NODE	Attribute	key = "value"
COMMENT_NODE	Comment	<--This is a comment-->
DOCUMENT_NODE	Document	The enclosing Document
ELEMENT_NODE	Element	<NAME>…</NAME>
TEXT_NODE	Body Text within element	Hello world

6.1.3. Java XML API - Xerces와 JDOM의 리용

Java와 XML을 함께 리용하는데 쓰이는 도구들은 아주 많다. 그중에서 가장 널리 리용되는것들은 `apache.org.`로부터 내리적재할수 있는 Xerces패키지와 `jdom.org.`로부터 얻을수 있는 JDOM패키지이다.

Xerces는 사실 SAX(판본2)규격은 물론 W3C XML과 DOM(수준 1과 2)표준들을

만족시키는 공인된 분석기들을 포함하고있다. Xerces는 크고 포괄적인 완전한 표준의 실현이다.

JDOM은 총체적으로 Java지향적인 방법으로 XML과 작업한다. JDOM은 XML자료를 읽고 쓰는 든든하고 가벼운 수단들을 제공한다. 작업하는데서는 Xerces API보다 더 직관적이지만 차이는 적다.

이책의 실례들에서 Xerces를 리용하는데 그 리유는 다음과 같다.

- Xerces는 DOM의 완전한 실현으로 예상되고있다.
- 실례코드들은 Xbeans.org.의 Xbeans에 기초하여 구성되였다. 원래의 Xbean코드는 Xerces를 리용한다.
- xerces.jar파일은 이 책의 실례들을 실현하는데 필요한 모든것들을 포함하고있다.

이미 말한바와 같이 실례코드들이 API의 극히 일부만을 리용하므로 실례들을 임의의 API로부터 다른 API로 변환하는것은 상대적으로 간단하다. 다음의 코드는 Xerces API를 리용하여 XML문서를 작성하는 방법을 보여준다.

```
Document doc = new DocumentImpl();
Element root = (Element)doc.createElement("SYSTEMDATE");
doc.appendChild(root);

Element year = (Element)doc.createElement("YEAR");
root.appendChild(year);
year.appendChild(doc.createTextNode(""+calendar.get(Calendar.YEAR)));
```

JDOM으로 작성한 실례는 우의 실례와 류사하며 Xerces실례보다 좀 더 간단하다. 그것은 JDOM이 XML과 함께 작업하는 Java지향적인 방법으로 설계되였기때문이다.

```
Element root = new Element("SYSTEMDATE");
Document doc = new Document(root);
Element year = new Element("YEAR");
root.addContent(year);
year.setText(""+calendar.get(Calendar.YEAR));
```

XML문서들을 다루는데서 DOM방법을 리용하는 기본 우점의 하나는 그 문서를 DOM객체로 분석하기만 하면 필요한 때 임의의 요소에 접근할수 있다는것이다.

6.1.4. Xbean을 XML처리블로크로 리용

Java에서 XML문서들과 작업하는데 가장 적합한 방법들중의 하나가 Xbean을 리용하는것이다. Xbean은 본질에 있어서 끼움가능한 XML처리블로크들이다. Xbean들은 사슬로 련결되며 이때 사슬의 매 Xbean은 XML문서를 처리하는데서 하나의 론리적단계를 수행하게 된다. 다음 Xbean은 문서를 bean사건의 사건객체로서 사슬고리의 다음 Xbean에 넘겨준다. 그림 6-2에서 이것을 보여준다.

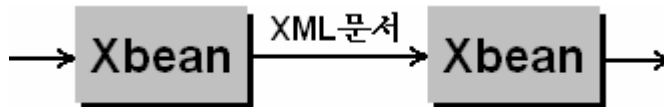



그림 6-2. Xbean의 접속성

접속성(connectivity)은 Xbean개념의 기본열쇠이다. 사슬의 매 Xbean은 한가지 처리단계를 수행하며 처리된 XML문서를 다음단계 처리를 위하여 다음 Xbean에 넘겨준다. 이런 방법을 리용하면 큰 프로젝트를 간단하고 반복적인 조작들로 쉽게 가를수 있으며 이때 매 조작들은 재리용가능한 구성요소로 실현될수 있다.

Xbean의 접속성은 다른 Xbean들과 통신하기 위한 위임사건모형(delegation event model)을 리용하여 실현된다. 위임사건모형은 DOM문서를 사건객체로 정의하는 DOMEvent대면부를 리용하여 실현된다. 사건원천인 Xbean은 DOMEvent를 일으킴으로써 XML문서를 DOMEvent객체로 EventListener Bean에 넘겨준다.

Xbean접속성모형은 org.Xbeans패키지안에 포함되여있는 두개의 대면부에 의하여 정의된다.

- DOMSource는 출력을 받는 Xbean을 설정하거나 얻기 위한 두개의 메쏘드 setDOMListener()와 getDOMListener()들을 정의한다.
- DOMListener는 한가지 메쏘드 documentReady(DOMEvent e)를 정의한다. 이 메쏘드는 사건원천인 Xbean에서 호출되며 DOMEvent속에 교갑화된 XML 문서를 넘겨준다.

일반적으로 doucumentReady()메쏘드는 XML문서를 처리하고 돌려주는 procssDocument(DOMEvent e)메쏘드를 호출한다. 만일 사슬에 또 다른 Xbean이 있다면 처리된 문서는 새로운 DOMEvent로 교갑화되여 사슬의 다음 Xbean에 넘겨진다.

실천에서 Xbean들을 리용하는 가장 간단한 방법은 인수가 없는 processDocument() 메쏘드로 기초클라스를 창조하는것이다. 그 다음 목적하는 기능을 수행하도록 processDocument()메쏘드를 재정의하여 Xbean기초클라스를 확장한다. 목록 6-2는 Xbean기초클라스를 보여준다.

목록 6-2. XBean기초클라스

```
package JavaDB_Bible.ch06.sec01.Xbeans;

import org.xbeans.*;
import org.w3c.dom.Document;

public class XBean implements org.xbeans.DOMListener,
   org.xbeans.DOMSource{
   protected DOMListener DOMListener;
   protected Document processedXmlDoc = null;

   public XBean(){
   }

   public void setDOMListener(DOMListener newDomListener){
      DOMListener = newDomListener;
   }

   public DOMListener getDOMListener(){
      return DOMListener;
   }

   public void documentReady(DOMEvent evt)
         throws XbeansException{
      processedXmlDoc = processDocument(evt.getDocument());
      if(DOMListener!=null)
         DOMListener.documentReady(new DOMEvent(this, processedXmlDoc));
   }

   public void processDocument() throws XbeansException{
   }

   public Document processDocument(Document doc)
         throws XbeansException {
      return doc;
   }
}
```

Xbean들은 보통 사슬을 리용하며 사슬에는 시작과 끝이 명백히 표시되여있다. 목록 6-3에서는 목록 6-2의 XBean기초클라스를 확장하여 쓸모있는 종결 Xbean을 창조하였다. 이 SerializerBean은 단순히 문서를 흐름(기정으로는 System.out)에 출력한다.

목록 6-3. SerializerBean

```
package JavaDB_Bible.ch06.sec01.Xbeans;

import java.io.*;
import org.xbeans.*;
import org.w3c.dom.Document;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;

public class SerializerBean extends
   JavaDB_Bible.ch06.sec01.Xbeans.XBean{
   protected OutputStream os = System.out;
   protected Writer writer = null;
   protected XMLSerializer serializer;

   public SerializerBean(){
   }

   public void setOutputStream(OutputStream os) {
      this.os = os;
   }

   public void setWriter(Writer writer){
      this.writer = writer;
   }

   public Document processDocument(Document doc){
      OutputFormat fmt = new OutputFormat("xml", null, true);
      if (writer != null){
         serializer = new XMLSerializer(writer, fmt);
      }
      else{
         serializer = new XMLSerializer(os, fmt);
      }

      if(doc!=null){
         try{
            serializer.asDOMSerializer().serialize(doc);
         }
         catch(Exception e){
            e.printStackTrace();
         }
      }
      return doc;
   }
}
```

Xbean사슬은 흐름으로부터 이미 존재하는 문서를 읽든가 혹은 어떤 자료원천으로부터 문서를 만드는 Xbean으로 시작한다. 목록 6-4는 체계시계에서 날자를 읽어들이고 XML문서를 창조하는 간단한 Xbean을 보여준다.

목록 6-4. SystemTimeBean

```
package JavaDB_Bible.ch06.sec01.Xbeans;

import java.io.*;
import java.util.Calendar;
import java.util.GregorianCalendar;
import org.xbeans.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.apache.xerces.dom.DocumentImpl;

public class SystemTimeBean extends JavaDB_Bible.ch06.sec01.Xbeans.XBean{
  protected String textFileName = "";

  public SystemTimeBean() {
  }

  public void processDocument() throws XbeansException{
    GregorianCalendar calendar = new GregorianCalendar();
    try{
      Document doc = new DocumentImpl();

      Element root = (Element)doc.createElement("SYSTEMDATE");
      doc.appendChild (root);

      Element year = (Element)doc.createElement("YEAR");
      root.appendChild(year);
      year.appendChild(doc.createTextNode("" +
                     calendar.get(Calendar.YEAR)));

      Element month = (Element)doc.createElement("MONTH");
      root.appendChild(month);
      int mon = calendar.get(Calendar.MONTH)+1;
      month.appendChild(doc.createTextNode(String.valueOf(mon)));

      Element day = (Element)doc.createElement("DAY");
```

```
      root.appendChild(day);
      int mDay = calendar.get(Calendar.DAY_OF_MONTH);
      day.appendChild(doc.createTextNode(String.valueOf(mDay)));

      Element dayOfWeek = (Element)doc.createElement("DAY_OF_WEEK");;
      root.appendChild(dayOfWeek);
      int wDay = calendar.get(Calendar.DAY_OF_WEEK);
      dayOfWeek.appendChild(doc.createTextNode(String.valueOf(wDay)));

      Element hour = (Element)doc.createElement("HOUR");;
      root.appendChild(hour);
      int h = calendar.get(Calendar.HOUR_OF_DAY);;
      hour.appendChild(doc.createTextNode(String.valueOf(h)));

      Element min = (Element)doc.createElement("MINUTE");
      root.appendChild(min);
      int m = calendar.get(Calendar.MINUTE);
      min.appendChild(doc.createTextNode(String.valueOf(m)));
      DOMListener.documentReady(new DOMEvent(this,doc));
    }catch (Exception e){
      throw(new XbeansException("","SystemTimeBean",
                                e.toString(),e.getMessage()));
    }
  }
}
```

이제는 사슬을 만들고 시험해볼만한 Xbean들이 충분히 마련되였다. 목록 6-5는 Xbean의 구체례들을 만들고 호상련결하는 방법을 설명하는 간단한 시험프로그람이다.

목록 6-5. Xbean들을 리용하여 XML문서를 출력

```
package JavaDB_Bible.ch06.sec01;

import java.io.*;
import java.beans.Beans;
import JavaDB_Bible.ch06.sec01.Xbeans.*;

public class SysTimeBeanTest {
   static public void main(String args[]){
     try{
```

```
        SystemTimeBean timeBean = (SystemTimeBean)Beans.instantiate(null,
          "JavaDB_Bible.ch06.sec01.Xbeans.SystemTimeBean");
        SerializerBean serializer = (SerializerBean)Beans.instantiate(null,
          "JavaDB_Bible.ch06.sec01.Xbeans.SerializerBean");
        timeBean.setDOMListener(serializer);
        serializer.setOutputStream(new
FileOutputStream("TimeStamp.xml"));
        timeBean.processDocument();
      }
      catch(Exception e){
        System.err.println(e);
      }
    }
}
```

목록 6-5의 실례에서 볼수 있는 바와 같이 Xbean들을 리용하여 XML문서들을 창조하고 처리하자면 다음 두 단계를 거쳐야 한다.

1. Xbean들의 구체례를 만들고 필요한 속성들을 설정한다.
2. 사슬의 첫번째 Xbean인 SystemTimeBean의 processDocument()메쏘드를 호출한다.

일어나는 사건렬들은 다음과 같다.

우선 SystemTimeBean은 다음의 코드를 리용하여 새로운 XML문서를 창조한다.

```
Document doc = new DocumentImpl();
```

그 다음 root요소를 창조하고 그것을 문서에 추가한다.

```
Element root=(Element)doc.createElement("SYSTEMDATE");
doc.appendChild(root);
```

그 다음에는 아래에서 보는바와 같이 년, 월, 일 등의 요소들을 창조하고 그것을 root요소에 추가한다.

```
Element root = (Element)doc.createElement("YEAR");
root.appendChild(year);
year.appendChild(doc.createTextNode("" + calendar.get(Calendar.YEAR)));
```

끝으로 SystemTimeBean은 자기의 등록된 감시자 SerializerBean의 documentReady()메쏘드를 호출하여 아래와 같이 새로운 XML문서를 넘겨준다.

```
DOMListener.documentReady(new DOMEvent(this, doc));
```

이번에는 SerializerBean이 자기의 outputStream속성에서 정의된 흐름에 문서를 직렬화하기 위하여 processDocument()메쏘드를 호출한다. 결과적인 XML을 목록 6-6에 보여주었다.

목록 6-6. Xbean을 리용하여 생성하고 직렬화한 XML날자도장

```
<?xml version="1.0"?>
<SYSTEMDATE>
   <YEAR>2007</YEAR>
   <MONTH>4</MONTH>
   <DAY>10</DAY>
   <DAY_OF_WEEK>7</DAY_OF_WEEK>
   <HOUR>16</HOUR>
   <MINUTE>54</MINUTE>
</SYSTEMDATE>
```

6.1.5. 자료기지질문에 의한 XML문서작성

자료기지로부터 XML문서들을 생성해내는것도 체계시계와 같은 원천으로부터 XML문서들을 만드는것과 마찬가지로 간단하다. 그림 6-1의 XML문서를 보고 알수 있는것처럼 그것은 자료기지의 표와 구조적으로 류사하다.

JDBC를 리용하여 XML문서들을 창조하는데는 기본적으로 두가지 방법이 있다. 첫번째는 5장 5절에서 자료기지로부터 자료를 추출하고 HTML로 출력하는데 JSP를 리용한것과 류사한 방법으로 JSP를 리용하여 자료를 추출하고 그것을 XML로 출력하는것이다. 다른 하나의 보다 융통성있는 방법은 XML문서를 직렬화하기전에 적절하게 처리할수 있는 문서의 DOM표현을 창조하는것이다.

SQL질문을 리용하여 DOM문서들을 창조하기 위해 설계된 Xbean을 목록 6-7에 보여주었다. 이 Xbean의 processDocument()메쏘드는 우선 사용할 자료기지와 표이름을 식별하는 뿌리요소를 가진 DOM문서를 창조한다. 그 다음 SQL질문을 수행하고 ResultSet의 매 행에 대응하는 요소들을 추가하는 appendDataNodes()메쏘드를 호출한다. 주문자번호가 이 요소에 속성으로 삽입된다.

요소들은 렬이름으로 설정된 태그이름들을 가지며 본문마디로 그 렬자료를 포함한다. 이 실례에서는 jdbc-odbc다리구동프로그람을 리용하고있지만 5장의 실례들에서 배운대로 DataSource코드를 쉽게 대입할수 있을것이다.

목록 6-7. SQL질문을 리용하여 XML문서를 작성

```
package JavaDB_Bible.ch06.sec01.Xbeans;

import java.io.*;
import java.sql.*;
import org.xbeans.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.apache.xerces.dom.DocumentImpl;

public class SQLQueryBean extends XBean{
   private String databaseName = "";
   private String tableName = "";
   private String SQLQuery = null;

   Document document;

   public SQLQueryBean(){
   }

   public void setDatabaseName(String databaseName){
      this.databaseName = databaseName;
   }

   public void setTableName(String tableName){
      this.tableName = tableName;
   }

   public void setSQLQuery(String SQLQuery){
      this.SQLQuery = SQLQuery;
   }

   public void processDocument() throws XbeansException{
      try{
         document = new DocumentImpl();
         if(databaseName.length()>0 && tableName.length()>0){
            String d = databaseName.toUpperCase();
            String t = tableName.toUpperCase();
            Element root = (Element)document.createElement(t);
            document.appendChild(root);
            root.setAttribute("DBNAME", d);
            appendDataNodes();
```

```
        }else{
            throw new XbeansException("SQLQueryBean",
                null, "DBName/TableName Undefined", null);
        }

        DOMEvent domEvt = new DOMEvent(this,document);
        DOMListener.documentReady(domEvt);
    }catch(Exception e){
        e.printStackTrace();
    }
}

public void appendDataNodes() {
    String url = "jdbc:odbc:"+databaseName;

    if(SQLQuery==null) SQLQuery = "SELECT * FROM "+tableName;
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(SQLQuery);
        ResultSetMetaData md = rs.getMetaData();
        int nColumns = md.getColumnCount();
        Element root = document.getDocumentElement();

        while(rs.next()){
            Element record = (Element)document.createElement("CUSTOMER");
            root.appendChild(record);

            for(int i=1;i<=nColumns;i++) {
                String fName = md.getColumnLabel(i);
                String data = rs.getString(i);
                if(fName.equals("Customer_ID")){
                    record.setAttribute("Customer_ID",
String.valueOf(data));
                }else{
                    Element fld = (Element)document.createElement(fName);
                    record.appendChild(fld);
                    fld.appendChild(document.createTextNode(data));
                }
            }
        }
```

```
            con.close();
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

목록 6-8에서 보여준 SQLQueryBeanTest코드는 목록 6-5에서 SystemTimeBean의 자리에 SQLQueryBean을 끼워넣었을뿐 다른것은 변경되지 않았다. 이것 역시 Xbean들을 리용하여 XML자료를 처리하는것이 얼마나 쉬운가를 보여준다.

목록 6-8. SQLQueryBean의 리용

```
package JavaDB_Bible.ch06.sec01;

import java.io.*;
import java.beans.Beans;
import JavaDB_Bible.ch06.sec01.Xbeans.*;

public class SQLQueryBeanTest{
    static public void main(String args[]){
        String databaseName = "SQLServerContacts";
        String tableName = "CUSTOMERS";
        String SQLQuery = "SELECT * FROM CUSTOMERS WHERE STATE = 'Pyongyang';";
        try{
            SQLQueryBean queryBean = (SQLQueryBean)Beans.instantiate(null,
                "JavaDB_Bible.ch06.sec01.Xbeans.SQLQueryBean");
            SerializerBean serializer = (SerializerBean)Beans.instantiate(null,
                "JavaDB_Bible.ch06.sec01.Xbeans.SerializerBean");
            queryBean.setDatabaseName(databaseName);
            queryBean.setTableName(tableName);
            queryBean.setSQLQuery(SQLQuery);
            queryBean.setDOMListener(serializer);
            serializer.setOutputStream(new
FileOutputStream("Customers.xml"));
            queryBean.processDocument();
        }catch(Exception e){
            System.err.println(e);
        }
    }
}
```

이 실례의 결과를 목록 6-9에 보여주었다. 독자들은 이것이 그림 6-1에 보여준 원래의 XML문서라는것을 알수 있을것이다.

목록 6-9. Customer표로부터 직렬화한 DOM문서

```
<?xml version="1.0"?>
<CUSTOMERS DBNAME="SQLSERVERCONTACTS">
  <CUSTOMER Customer_ID="100">
    <Family_Name>김</Family_Name>
    <Personal_Name>성철</Personal_Name>
    <Sex>남</Sex>
    <State>평양시</State>
    <City>평양</City>
    <District>중구역</District>
    <Dong>교구동</Dong>
    <Neighbor>32</Neighbor>
    <Phone>352-6532</Phone>
  </CUSTOMER>
  <CUSTOMER Customer_ID="101">
    <Family_Name>김</Family_Name>
    <Personal_Name>은희</Personal_Name>
    <Sex>녀</Sex>
    <State>평양시</State>
    <City>평양</City>
    <District>보통강구역</District>
    <Dong>대보동</Dong>
    <Neighbor>17</Neighbor>
    <Phone>453-5521</Phone>
  </CUSTOMER>
  <CUSTOMER Customer_ID="104">
    <Family_Name>강</Family_Name>
    <Personal_Name>권혁</Personal_Name>
    <Sex>남</Sex>
    <State>평양시</State>
    <City>평양</City>
    <District>중구역</District>
    <Dong>류성동</Dong>
    <Neighbor>23</Neighbor>
    <Phone>321-8710</Phone>
  </CUSTOMER>
  <CUSTOMER Customer_ID="108">
    <Family_Name>오</Family_Name>
```

```
    <Personal_Name>성철</Personal_Name>
    <Sex>남</Sex>
    <State>평양시</State>
    <City>평양</City>
    <District>평천구역</District>
    <Dong>안산2동</Dong>
    <Neighbor>56</Neighbor>
    <Phone></Phone>
</CUSTOMER>
</CUSTOMERS>
```

자료기지로부터 XML을 생성할수 있은것처럼 XML자료를 자료기지에 써넣을수도 있다. 인터네트에서는 이런 방법으로 리용할수 있는 많은 XML자료원천들을 제공한다.

6.1.6. XML자료원천들을 리용하여 자료기지에 자료써넣기

XML이 현저하게 상승하게 된 요인은 인터네트를 통하여 접근할수 있는 XML기초의 봉사들이 수많이 개발되였기때문이다. 이러한 봉사의 우수한 실례는 Moreover.com에 의한 XML기초의 새소식봉사이다. 목록 6-10은 다음의 주소에서 리용할수 있는 인기기사제목련결페지의 형식을 설명한다.

```
http://www.moreover.com/cgi-local/page?o=xml&query=top+stories.
```

XML문서는 뿌리태그 <moreovernews>를 포함하고 뿌리태그는 많은 <article>요소를 포함한다. 이 매개 요소는 11개의 자식요소를 포함한다. <article>요소들과 그것들의 자식요소들은 함께 11개의 자식요소들에 대응하는 11개의 렬을 가진 자료기지표의 행으로 간주될수 있다.

목록 6-10. Moreover.com의 인기기사제목 XML

```
<?xml version= "1.0" encoding="iso-8859-l"?>
<!DOCTYPE moreovernews SYSTEM
"http://p.moreover.com/xml_dtds/moreovernews.dtd">
<!-- by using this feed you have read and agree to our terms and conditions
at http://w.moreover.com/site/about/termsandconditions.html
If the presence of this comment has caused an error in your parser
you may use the older uncommented version by using &amp;o=xml_1 or
+xml_1 in the URL.
Using the xtnl_1 version still means that you have read and agree to
our terms and conditions above -->
```

```
<moreovernews>
  <article id="_34226715">
    <url>http ://c.moreover.com/click/here.pl?x34226715</url>
    <headline_text>
      Assistant fire chief ascends to departments top role
    </headline_text>
    <source>Springfield News-Leader</source>
    <media_type>text</media_type>
    <cluster>moreover...</cluster>
    <tagline></tagline>
    <document_url>
      http://www.springfieldnews-leader.com/news/
    </document_url>
    <harvest_time>Mar 20 2002 2:30AM</harvest_time>
    <access_registration></access_registration>
    <access_status></access_status>
  </article>
  ...
</moreovernews>
```

자료기지표에 XML문서내용을 삽입하는데 리용되는 SQLInsertBean은 목록 6-2의 Xbean기초클라스를 확장한것이다. processDocument()메쏘드는 먼저 SQL의 INSERT 지령을 처리하는 PreparedStatement를 창조하는 prepareStatement()메쏘드를 호출한다. PreparedStatement는 단순히 article요소(10개의 자식요소와 id속성)의 매 자료마디에 하나씩 해당되는 11개의 자리유지자를 가진 SQL INSERT지령이다.

getValues()메쏘드는 id속성과 그 자식요소들에 대한 검색을 위해 분석해야 할 <article>요소를 넘겨받는다. id속성과 그 자식요소들은 String배렬로 반환되며 배렬값은 insertHeadline()메쏘드에 넘겨진다. 특히 Bean 자식요소들에 대해서는 Java의 Null값이 배렬속에 삽입된다. 이 Null값들은 삽입될 때 자동적으로 SQL의 Null값으로 변환된다.

insertHeadline()메쏘드는 String배렬로부터 PreparedStatement의 파라메터들을 설정한 다음 XML문서로부터 자료를 삽입하기 위하여 PreparedStatement의 executeUpdate()메쏘드를 호출한다.

모든 <article>요소들을 다 순환한 다음에는 Connection객체의 close()메쏘드가 호출되여 DataSource에 대한 접속을 끝낸다. 자료기지에 새소식표제들을 삽입하기 위한 PreparedStatement객체의 리용방법을 목록 6-11에 보여주었다.

목록 6-11. SQLInsertBean

```
package JavaDB_Bible.ch06.sec01;

import java.io.*;
import java.sql.*;
import javax.sql.*;
import org.xbeans.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import JavaDB_Bible.ch06.sec01.Xbeans.*;

public class SQLInsertBean extends JavaDB_Bible.ch06.sec01.Xbeans.XBean{
   static String moreoverUrl =

"http://www.moreover.com/cgi-local/page?o=xml&query=top+stories";
   private static String dbUserName = "sa";
   private static String dbPassword = "dba";
   Connection con = null;
   PreparedStatement pstmt = null;

   public SQLInsertBean() {
   }

   public Document processDocument(Document doc) throws XbeansException
{
      prepareStatement();
      Element root = doc.getDocumentElement();
      NodeList articles = root.getElementsByTagName("article");
      for(int i=0;i<articles.getLength();i++){
         Element article = (Element)articles.item(i);
         insertHeadline(getValues(article));
      }
      closeConnection();
      return doc;
   }

   private String[] getValues(Element article){
      String[] values = new String[11];
      values[0] = article.getAttribute("id");
      NodeList dataNodes = article.getChildNodes();
```

```
    for(int i=0,j=1;i<dataNodes.getLength();i++){
      Node dataNode = dataNodes.item(i);
      if(dataNode.getNodeType()==Node.ELEMENT_NODE){
        Node textNode = ((Element)dataNode).getFirstChild();
        if(textNode!=null) values[j++] = textNode.getNodeValue();
        else values[j++] = null;

      }
    }
    return values;
  }

  private void prepareStatement(){
    try {
      Class.forName("com.inet.pool.PoolDriver");
      com.inet.tds.TdsDataSource tds = new com.inet.tds.TdsDataSource();
      tds.setServerName( "DOLPHIN" );
      tds.setDatabaseName( "MOREOVERNEWS" );
      tds.setUser( dbUserName );
      tds.setPassword( dbPassword );

      DataSource ds = tds;
      con = ds.getConnection(dbUserName,dbPassword);
      String SQLCmd = "INSERT INTO HEADLINES Values "+
                     "(?,?,?,?,?,?,?,?,?,?,?);";
      pstmt = con.prepareStatement (SQLCmd);
    }
    catch(ClassNotFoundException e){
      System.err.println(e.getMessage());
    }
    catch(SQLException e) {
      System.err.println(e.getMessage());
    }
  }

  private void closeConnection(){
    try{
      con.close();
    }
    catch(SQLException e){
      System.err.println(e.getMessage());
    }
```

```
   }

   private int insertHeadline(String [] values) {
      int rowsInserted = -1;
      try {
         for (int i=0;i<values.length;i++) {
            pstmt.setString(i+1, fixApostrophes(values[i]));
         }
         rowsInserted = pstmt.executeUpdate();
      }catch(SQLException e){
         System.err.println(e.getMessage());
      }
      return rowsInserted;
   }

   private String fixApostrophes(String in){
      if (in!=null){
         int n=0;
         while ((n=in.indexOf("'", n))>=0){
            in = in.substring(0, n) + "'" + in.substring(n);
            n+=2;
         }
      }
      return in;
   }

   public  static void main(String[] args) {
      try{
         DOMParserBean parser = new DOMParserBean();
         SQLInsertBean insertBean = new SQLInsertBean();
         SerializerBean serializer = new SerializerBean();

         parser.setUrlString(moreoverUrl);
         parser.setDOMListener(insertBean);
         //insertBean.setDOMListener(serializer);

         parser.processDocument();
      }catch(Exception e){
         System.err.println("Exception in SQLInsertBeanTest");
      }
   }
}
```

목록 6-11의 실례에서 참조된 DOMParserBean을 목록 6-12에 보여주었다. 다시 한번 더 말하지만 이 클라스는 XBean기초클라스의 단순한 확장이다. 그의 getXml()메쏘드는 파일 혹은 URL로부터 읽어들였거나 단순히 String으로 넘겨진 XML문서를 포함하는 바이트배렬을 돌려준다. processDocument()메쏘드는 이 바이트배렬을 문서의 DOM표현으로 변환하기 위하여 Xerces DOMParser를 리용한다. 그 다음 DOM문서는 DOMEvent를 리용하여 XBean기초클라스에서 정의된 DOMListener에 넘겨진다.

목록 6-12: DOMParserBean

```
package JavaDB_Bible.ch06.sec01.Xbeans;

import java.io.*;
import java.net.*;
import org.xbeans.*;
import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import org.apache.xerces.parsers.DOMParser;

public class DOMParserBean extends JavaDB_Bible.ch06.sec01.Xbeans.XBean{
   protected String UrlString = null;
   protected String XmlString = null;
   protected String XmlFileName = null;

   public DOMParserBean(){
   }

   public void setXmlString(String XmlString){
      this.XmlString = XmlString;
   }

   public void setUrlString(String UrlString){
      this.UrlString = UrlString;
   }

   public void setXmlFileName(String XmlFileName){
      this.XmlFileName = XmlFileName;
   }

   public void processDocument(){
      try{
         byte[] xml  = getXml();
         if (xml.length > 0) {
            ByteArrayInputStream x = new ByteArrayInputStream(xml);
            DOMParser p = new DOMParser();
```

```
            InputSource s = new InputSource(x);
            p.parse(s);
            Document doc = p.getDocument();
            DOMEvent e = new DOMEvent(this,doc);
            DOMListener.documentReady(e);
         }
      }
      catch (Exception e){
         e.printStackTrace();
      }
   }

   private byte[] getXml(){
      byte[] xml = new byte[4096];
      if(XmlFileName!=null){
         File f = new File(XmlFileName);
         xml = new byte[(int)f.length()];
         try{
            FileInputStream is = new FileInputStream(f);
            is.read(xml,0,(int)f.length());
         }
         catch (Exception e) {
            e.printStackTrace();
         }
      }else if(UrlString!=null){
         URL fileURL = null;
         String buffer = "";
         try {
            fileURL = new URL(UrlString);
            InputStream inputstream = fileURL.openStream();
            int byteCount;
            while ((byteCount = inputstream.read(xml))>0){
               buffer += new String (xml, 0, byteCount);
            }
            xml = buffer.getBytes();
         }
         catch (Exception e){
            e.printStackTrace();
         }
      }else {
         if (XmlString!=null) xml = XmlString.getBytes();
      }
      return xml;
   }
}
```

운반기구로서의 XML과 저장매체로서의 관계형자료기지의 결합은 웨브기반의 구성방식으로 더욱더 보편화될것이다. 이 두 기술들은 서로 상대측에 없는 능력을 제공하면서 호상보완하고있다.

제2절. RowSet를 리용한 자료의 현시

RowSet는 JDBC API에 JavaBean의 지원을 추가함으로써 JDBC에 새롭고 중요한 능력을 보충하였다. RowSet를 리용하면 망상에서 표자료를 쉽게 전송할수 있다. 또한 밑준위에 있는 JDBC구동프로그람이 ResultSet들을 지원하지 않는 경우 흘리기가능한 ResultSet나 갱신가능한 ResultSet들을 제공하는 포장제로 리용할수 있다.

이 절에서는 RowSet와 JDBC 핵심 API의 ResultSet를 비교하고 여러가지 류형의 RowSet들에 대한 특징을 설명한다.

6.2.1. RowSet에 대한 리해

RowSet는 ResultSet 혹은 파일이나 펼친표와 같이 표로 된 기타 자료원천으로부터 얻은 행들의 모임을 포함하고있는 객체이다. 다시 말하여 RowSet객체는 JavaBeans의 지원을 협력하기 위한 목적으로 추가된 ResultSet의 확장이다. RowSet객체는 ResultSetMetaData대면부를 확장한 RowSetMetaData대면부에 의하여 지원된다.

RowSet는 자료원천에 접속하고 지령을 실행하며 자료원천으로부터 자료를 읽거나 쓰기 위하여 JDBC자료원천에 대한 접속과 그 자료원천으로부터 자료를 읽기 위한 JavaBeans속성모임을 제공한다는 점에서 ResultSet와 중요하게 구별된다.

그러한 속성들은 다음과 같다.

- rowSet.setUrl(url);
- rowSet.setUsername(login);
- rowSet.setPassword(password);
- rowSet.getConnection();
- rowSet.setCommand("SELECT * FROM sysusers");
- rowSet.execute();

RowSet는 JavaBean이므로 사용자이름과 통과암호와 같은 속성들을 설정하고 그 값을 얻는데서 JavaBean모형에 따른다. 또한 RowSet는 렬값의 변화와 같은 사건들을 처리하기 위하여 JavaBeans API를 계승한다.

RowSet도 ResultSet와 마찬가지로 접속형과 비접속형이 있다. 접속형 RowSet는 사용중에 있는 동안 자료원천에 대한 접속을 계속 유지하지만 비접속형 RowSet는 자료를

적재하거나 변화된 내용을 자료원천에 반영할 때에만 접속을 열고 기타 기간에는 접속을 닫는다.

RowSet의 만들기와 리용

실례를 통하여 RowSet가 어떻게 동작하는가를 알아보자. 목록 6-13은 1장에서 작성한 주문자표(Customers)에서 주문자들의 이름과 전화번호를 JdbcRowSet를 리용하여 얻어내는 실례이다.

실례에서는 RowSet의 메쏘드들에 중심을 두고있다. 만일 ResultSet로 작업을 한다면 다음과 같은 객체들을 만들어야 할것이다.

- java.sql.Connection
- java.sql.Statement
- java.sql.ResultSet

RowSet를 리용할 때에는 RowSet자체의 필요한 속성들을 설정한다. 그리고 RowSet.execute()메쏘드를 리용하여 SQL지령을 수행한다. JdbcRowSet는 ResultSet를 JavaBeans구성요소로서 리용할수 있게 하는 ResultSet객체의 포장제로 실현되였다.

JdbcRowSet는 JDBC구동프로그람을 리용하여 자료기지에 대한 접속을 유지하는 접속형 RowSet이므로 그 구동프로그람을 JavaBeans부분품으로 효과적으로 만들수 있다.

목록 6-13. RowSet의 리용

```
package JavaDB_Bible.ch06.sec02;

import java.sql.*;
import com.sun.rowset.JDBCRowSetImpl;

public class JDBCRowSetExam1{
  public static void main (String[] argv){
    String url = "jdbc:inetdae7:localhost:1433?database=CUSTOMERS";
    String login = "sa";
    String password = "dba";
    try {
      Class.forName("com.inet.tds.TdsDriver").newInstance();
      JdbcRowSetImpl rowSet = new JdbcRowSetImpl();

      rowSet.setUrl( url );
      rowSet.setUsername( login );
      rowSet.setPassword( password );
```

```
     DatabaseMetaData dbmd = rowSet.getConnection().getMetaData();
     System.out.println("Driver Name:\t" + dbmd.getDriverName());
     System.out.println("Driver Version:\t" + dbmd.getDriverVersion());

     rowSet.setCommand("SELECT
Customer_ID,Family_Name,Personal_Name,"+
                    "Phone FROM CUSTOMERS");

     rowSet.execute();

     while (rowSet.next()){
       for(int j=1;j<=rowSet.getMetaData().getColumnCount();j++){
         System.out.print(rowSet.getObject(j)+" \t");
       }
       System.out.println();
     }
     rowSet.close();
   }
   catch(Exception e){
     e.printStackTrace();
   }
  }
}
```

목록 6-13의 결과를 표 6-2에 보여준다.

표 6-2. JDBCRowSetExample의 결과

ID	Family_Name	Personal_Name	Phone
100	김	성철	326-6532
101	김	은희	453-5521
102	김	철	
103	박	성호	
104	강	권혁	321-8710
105	하	성진	
106	최	영실	
107	리	성민	
108	오	성철	

흘리기 및 갱신가능한 RowSet 만들기

JDBC Extension API에는 ResultSet를 흘리기 및 갱신가능한것으로 만드는 기능이 있다. RowSet에도 적당한 속성들을 설정하면 그러한 능력을 추가할수 있다.

목록 6-14는 목록 6-13에서 만든 RowSet를 흘리기가능하게 하는 간단한 방법을 보여준다.

목록 6-14. RowSet를 흘리기가능한것으로 만들기

```
package JavaDB_Bible.ch06.sec02;

import java.sql.*;
import com.inet.tds.JDBCRowSet;

public class JDBCRowSetExam2{
  public static void main(String[] argv) {
    String url = "jdbc:inetdae7:localhost:1433?database=CUSTOMERS";
    String login = "sa";
    String password = "dba";

    try{
      Class.forName("com.inet.tds.TdsDriver").newInstance();
      JDBCRowSet rowSet = new JDBCRowSet();

      rowSet.setUrl( url );
      rowSet.setUsername( login );
      rowSet.setPassword( password );

      rowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);

      rowSet.setCommand("SELECT Customer_ID, Family_Name, " +
                        "Personal_Name, Phone FROM CUSTOMERS");
      rowSet.execute();

      while(rowSet.next()){
        for(int j=1; j<=rowSet.getMetaData().getColumnCount();j++){
           System.out.print(rowSet.getObject(j)+"\t");
        }
        System.out.println();
      }

      while(rowSet.previous()){
```

```
        for(int j=1;j<=rowSet.getMetaData().getColumnCount();j++){
          System.out.print(rowSet.getObject(j)+"\t");
        }
        System.out.println();
      }
      rowSet.close();
    }catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

목록 6-14의 결과는 RowSet.previous()메쏘드를 리용하여 행들을 거꾸로 다시한번 출력한것을 제외하고는 목록 6-13의 결과와 같다. previous()메쏘드외에도 ResultSet로부터 계승한 다른 유표조종메쏘드들과 흘리기가능하게 하는 메쏘드들을 리용할수 있다.

RowSet를 갱신가능하게 하는 방법도 우와 같다. 갱신가능한 RowSet는 RowSet자체내에서 값들을 갱신할수 있다. 이때 변경된 내용은 RowSet.updateRow()메쏘드가 호출된 후에 자료기지에 반영된다.

RowSet를 갱신가능하게 하려면 RowSet의 Concurrency속성을 ResultSet.CONCUR_UPDATABLE로 설정하면 된다.

```
rowSet.setConcurrency(ResultSet.CONCUR_UPDATABLE);
```

갱신가능한 RowSet를 리용하면 새로운 행을 삽입할수도 있고 이미 있던 행을 지울수 있으며 렬값들을 변경할수도 있다.

갱신가능한 RowSet을 요구한다고 하여 그의 갱신가능성이 실제로 얻어진다는것은 담보되지 않는다. 때문에 RowSet.getConcurrency()메쏘드를 리용하여 RowSet가 실지로 갱신가능한가를 검사하여야 한다.

목록 6-15는 RowSet를 갱신가능하게 만들고 그것이 실지로 갱신가능한가를 확인하는 실례이다.

목록 6-15. RowSet를 갱신가능한것으로 만들기

```
package JavaDB_Bible.ch06.sec02;

import java.sql.*;
import com.inet.tds.JDBCRowSet;
```

```
public class JDBCUpdatableRowSet{
  public static void main(String[] argv){
    String url = "jdbc:inetdae7:localhost:1433?database=CUSTOMERS";
    String login = "sa";
    String password = "dba";

    try{
      Class.forName("com.inet.tds.TdsDriver").newInstance();
      JDBCRowSet rowSet = new JDBCRowSet();

      rowSet.setUrl( url );
      rowSet.setUsername( login );
      rowSet.setPassword( password );

      rowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
      rowSet.setConcurrency(ResultSet.CONCUR_UPDATABLE);

      rowSet.setCommand("SELECT Customer_ID,Family_Name,Personal_Name," +
                    "Phone FROM CUSTOMERS WHERE Family_Name = '김'");

      rowSet.execute();

      if (rowSet.getConcurrency() == ResultSet.CONCUR_UPDATABLE)
        System.out.println("Rowset is UPDATABLE");
      else
        System.out.println("Rowset is READ_ONLY");

      while (rowSet.next()){
        rowSet.updateString("Family_Name", "리");
        rowSet.updateRow();
        for(int j=1;j<=rowSet.getMetaData().getColumnCount();j++){
          System.out.print(rowSet.getObject(j)+"\t");
        }
        System.out.println();
      }
      rowSet.close();
    }
    catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

RowSet의 갱신

목록 6-15에서 알수 있는것처럼 갱신가능한 RowSet의 리용은 습관적인 SQL의 UPDATE지령을 리용하는것보다 더 간단하다. 이것은 특히 갱신가능한 RowSet에 준 변화가 항상 현재 행에 반영되기때문에 갱신한 행을 찾을 필요가 없을 때 효과가 있다. 물론 이것은 자료를 갱신하기전에 정확한 행으로 유표를 이동해야 한다는것을 의미한다.

RowSet의 갱신은 ResultSet로부터 계승된 갱신메쏘드들을 리용한다. 대부분의 ResultSet.update메쏘드들은 두개의 파라메터 즉 갱신할 렬과 그 렬에 넣을 새 값을 가진다. 이때 렬은 렬이름이나 렬번호를 리용하여 지적할수 있다. 표 6-3에서는 자료형에 따르는 ResultSet의 갱신메쏘드들을 보여주었다.

표 6-3. ResultSet의 Update메쏘드들

Data Type	Method
BigDecimal	updateBigDecimal(String columnName, BigDecimal x)
boolean	updateBoolean(String columnName, boolean x)
byte	updateByte(String columnName, byte x)
byte[]	updateBytes(String columnName, byte[] x)
double	updateDouble(Strlng columnName, double x)
float	updateFloat(String columnName, float x)
int	updateInt(String columnName, int x)
java.io.InputStream	updateAsciiStream(String columnName, InputStream x, int length)
java.io.InputStream	updateUnicodeStream(String columnName, InputStream x, int length)
java.io.InputStream	updateBinaryStream(String columnName, InputStream x, int length)
java.sql.Date	updateDate(String columnName, Date x)
java.sql.Time	updateTime(String columnName, Time x)
java.sql.Timestamp	updateTimestamp(String columnName, Timestamp x)
long	updateLong(String columnName, long x)
Object	updateObject(String columnName, Object x)
Object	updateObject(String columnName, Object x, int scale)
short	updateShort(String columnName, short x)
String	updateString(String columnName, String x)
NULL	updateNull(String columnName)

경고: 렬값을 갱신한 다음에는 유표를 이동하기전에 자료기지에 변화된 내용을 반영하는 updateRow()메쏘드를 호출해야 한다. 왜냐하면 갱신메쏘드를 리용하여 변경한 내용들은 updateRow()메쏘드를 호출하기전까지는 자료기지에 반영되지 않기때문이다. updateRow()를 호출하기전에 다른 행으로 유표를 이동시키면 변경했던 내용은 없어지고 행은 이전의 렬값으로 다시 변한다.

새로운 행의 삽입과 삭제

갱신가능한 RowSet는 자료를 변경하는것과 함께 전체 행의 삽입과 삭제도 지원해준다. 갱신가능한 RowSet객체는 ResultSet로부터 삽입행을 계승하였는데 이것은 사실상 새로운 행을 구축할수 있는 전용의 행완충기이다.

새로운 행의 삽입은 다음의 단계들로 이루어진다.

- moveToInsertRow()메쏘드를 호출하여 행을 삽입할 위치로 유표를 이동한다.
- 해당한 갱신메쏘드를 리용하여 그 행의 매 렬에 새로운 값을 설정한다.
- 새로운 행을 RowSet와 동시에 자료기지에 삽입하기 위하여 insertRow메쏘드를 호출한다.

목록 6-16에서는 갱신가능한 RowSet를 리용하여 자료기지에 새로운 행을 삽입하는 실례를 보여준다.

목록 6-16. 갱신가능한 RowSet를 리용한 새로운 행의 삽입

```
package JavaDB_Bible.ch06.sec02;

import java.sql.*;
import javax.sql.*;
import com.inet.tds.JDBCRowSet;

public class JDBCUpdatableRowSetInsert{
  public static void main(String[] argv){
  String url = "jdbc:inetdae7:localhost:1433?database=CUSTOMERS";
  String login = "sa";
  String password = "dba";

  try{
    Class.forName("com.inet.tds.TdsDriver").newInstance();
    JDBCRowSet rowSet= new JDBCRowSet();

    rowSet.setUrl (url );
    rowSet.setUsername( login ) ;
```

```
      rowSet.setPassword( password );

      rowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
      rowSet.setConcurrency(ResultSet.CONCUR_UPDATABLE);

      rowSet.setCommand("SELECT * FROM CUSTOMERS");

      rowSet.execute();
      if (rowSet.getConcurrency()==ResultSet.CONCUR_UPDATABLE)
        System.out.println("Rowset is UPDATABLE");
      else
        System.out.println("Rowset is READ_ONLY");

      rowSet.moveToInsertRow();

      rowSet.updateInt("Customer_ID", 115);
      rowSet.updateString("Family_Name", "여");
      rowSet.updateString("Pesonal_Name", "은하");
      rowSet.updateString("Sex", "녀");
      rowSet.updateString("State", "평양시");
      rowSet.updateString("City", "평양");
      rowSet.updateString("District", "평천구역");
      rowSet.updateString("Dong", "봉남동");
      rowSet.updateString("Neighbor", "29");
      rowSet.updateString("Phone", "456-0123");

      rowSet.insertRow();

      rowSet.beforeFirst();
      while (rowSet.next()){
        for(int j=1;j<=rowSet.getMetaData().getColumnCount();j++){
          System.out.print(rowSet.getObject(j)+"\t");
        }
        System.out.println();
      }
      rowSet.close();
    }catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

만일 새로운 행을 삽입하면서 그 행의 매 렬에 값을 주지 않는 경우 그 렬에 기정값이 있으면 기정값이 설정된다. 렬값을 지적하지 않은 경우 만일 그 렬이 SQL의 NULL값을 허용한다면 NULL값이 삽입된다. 만일 새 값도 주지 않고 NULL값도 허용하지 않는다면 SQL례외가 발생한다.

갱신가능한 RowSet에서 행을 삭제하려면 지우려는 행에 유표를 이동하고 deleteRow()메쏘드를 호출한다. 다음의 코드는 ResultSet에서 유표를 세번째 행으로 이동시키고 deleteRow()메쏘드를 써서 그 행을 지우는 방법을 보여준다.

```
rowSet.absolute(3);
rowSet.deleteRow();
```

갱신가능한 RowSet에서 일어난 변화보기

갱신가능한 RowSet에서 일어난 변화가 RowSet자체나 기타 조작중의 거래업무에서 반드시 보이는것은 아니다. 응용프로그람에서는 적당한 DatabaseMetaData메쏘드들을 호출하여 ResultSet에서 일어난 변화가 ResultSet자체에 보이는지 안보이는지를 확인할수 있다.

표에서 가장 최근의 자료를 얻으려면 목적하는 행으로 유표를 이동하고 refreshRow()를 호출하면 된다.

```
rs.absolute(3);
rs.refreshRow();
```

주의: 이때 RowSet는 TYPE_SCROLL_SENSITIVE로 되여야 하며 만약 그렇지 않으면 refreshRow()메쏘드는 아무런 작용도 하지 않는다.

RowSet의 사건

RowSet의 사건들은 RowSet에서 렬값의 변화와 같이 어떤 사건이 일어났을 때 발생한다. RowSet는 JavaBeans이므로 RowSet의 변화를 감시자에게 통지하는데 Java사건모형을 리용할수 있다.

RowSetListner메쏘드들은 다음과 같다.

- rowSetChanged－RowSet가 변경되였을 때 실례로 SQL지령이 수행되였을 때 호출된다.
- rowChanged － 행이 삽입, 갱신 혹은 삭제되였을 때 호출된다.
- cursorMoved － RowSet의 유표가 이동하였을 때 호출된다.

목록 6-17에서는 RowSet의 변화를 감시하는데 리용하는 RowSet의 사건들을 보여준다. 이 실례에서 RowSetListner는 CUSTOMERS표에 새로운 행이 삽입되였다는것을 통지하기

위하여 리용되였다. RowSetListner에서 RowSet.moveToCurrentRow()메쏘드의 리용에 주의를 돌려야 한다. 이 메쏘드는 유표가 삽입행에 있을 때 현재 행을 돌려주는데 리용된다. 이것은 RowSetListner가 방금 추가된 행의 내용을 통지할수 있게 한다.

목록 6-17. RowSet의 사건의 리용

```
package JavaDB_Bible.ch06.sec02;

import java.sql.*;
import javax.sql.*;
import com.inet.tds.JDBCRowSet;

public class JDBCUpdatableRowSetMonitor{
   public static void main(String[] argv){
      String url = "jdbc:inetdae7:localhost:1433?database=CUSTOMERS";
      String login = "sa";
      String password = "dba";
      try{
         Class.forName("com.inet.tds.TdsDriver").newInstance();
         JDBCRowSet rowSet = new JDBCRowSet();

         rowSet.setUrl( url );
         rowSet.setUsername( login );
         rowSet.setPassword( password );

         rowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
         rowSet.setConcurrency(ResultSet.CONCUR_UPDATABLE);

         rowSet.addRowSetListener(new RowSetChangeListener());

         rowSet.setCommand("SELECT * FROM CUSTOMERS");

         rowSet.execute();
         rowSet.moveToInsertRow();

         rowSet.updateInt("Customer_ID", 115);
         rowSet.updateString("Family_Name", "여");
         rowSet.updateString("Pesonal_Name", "은하");
         rowSet.updateString("Sex", "녀");
         rowSet.updateString("State", "평양시");
         rowSet.updateString("City", "평양");
         rowSet.updateString("District", "평천구역");
```

```
        rowSet.updateString("Dong", "봉남동");
        rowSet.updateString("Neighbor", "29");
        rowSet.updateString("Phone", "456-0123");

        rowSet.insertRow();
        rowSet.close();
      }
      catch (Exception e){
      }
   }
}

class RowSetChangeListener implements RowSetListener{
   public void rowSetChanged(RowSetEvent event){
   }

   public void rowChanged(RowSetEvent event){
      RowSet rowSet = (RowSet)event.getSource();
      try{
         rowSet.moveToCurrentRow();
         for(int j=1;j<=rowSet.getMetaData().getColumnCount();j++) {
            System.out.print(rowSet.getObject(j)+"\t");
         }
         System.out.println();
      }
      catch(Exception e){
      }
   }
   public void cursorMoved(RowSetEvent event) {
      RowSet rowSet = (RowSet)event.getSource();
      try{
         System.out.println("cursor moved to row "+rowSet.getRow());
      }
      catch(Exception e){
      }
   }
}
```

RowSet의 멋드러진 특징의 하나는 그것을 ResultSet처럼 리용할수 있는 외에 자료기지와 분리된 자료포함기로 리용할수 있다는것이다. 이에 대해서는 다음 소절에서 본다.

6.2.2. 비접속형 RowSet

이미 언급한것처럼 접속형 RowSet가 리용중에 있는 전기간 자료기지에 대한 접속을 유지한다면 비접속형 RowSet는 필요한 때에만 자료원천에 접속한다. 현재 많은 비접속형 RowSet의 실현들이 발표되여있다. 대표적으로 다음과 같은것들을 들수 있다.

- CachedRowSet
- JdbcRowSet
- WebRowSet

CachedRowSet는 ResultSet객체의 행들을 기억기에 완충기억시키며 따라서 자료기지에 대한 계속적인 접속을 필요로 하지 않는다. CachedRowSet들은 흘리기 및 갱신가능하다.

비접속이라는 의미는 CachedRowSet가 행들을 적재하기 위하여 자료를 읽을 때와 일어난 변화를 기초자료기지에 반영할 때에만 자료원천에 접속한다는것을 의미한다. 그 나머지시간에는 RowSet에서 변경이 진행되고있는 동안에조차 자료원천에 접속하지 않는다. 사실상 CachedRowSet객체는 단순히 JaveBean에 완충된 비접속형 행들의 모임으로 볼수 있다.

CachedRowSet객체의 갱신은 JdbcRowSet의 갱신과 류사하다. 그러나 RowSet가 갱신되고있는 동안에도 자료원천에 접속하지 않기때문에 기초자료원천에 변화를 주기 위한 추가적인 단계가 필요하다. 다시말하여 CachedRowSet객체는 updateRow()메쏘드나 insertRow()메쏘드를 호출한 다음 갱신된 내용을 자료원천에 써넣기 위하여 acceptChanges()메쏘드를 호출해야 한다.

PDA와 CachedRowSet의 리용

CachedRowSet는 JavaBean이므로 다른 임의의 JavaBean들처럼 직렬화될수 있다. 이것은 PDA(Personal Digital Assistant)와 같은 원격의뢰기와 작업할 때 아주 효과적으로 리용할수 있다. CachedRowSet는 가볍고 직렬화가능한것으로 하여 무선으로도 쉽게 전송할수 있으며 PDA와 같이 빈약한 의뢰기에 자료를 전송하는데 널리 리용된다.

이미 앞에서 본 주문자체계에서 주문자들이 상품을 주문하면 상점에서는 제날자에 손님들에게 상품을 수송해주어야 한다. 이를 위해서는 상품조달원들이 자기들의 PDA에 해당 날자에 따르는 주문자들의 정보를 복사해두어야 한다. 이것을 실현하기 위한 훌륭한 방도가 CachedRowSet의 리용이다. 그것은 CachedRowSet가 자료를 읽거나 갱신할 때에만 자료원천에 접속하기때문이다.

필요한 주문자들에 대한 정보를 얻는데서 처음으로 고려할것은 RowSet를 구축하는데 필요한 SQL질문이다. 여기서의 실례에서 필요한 주문자정보는 아래와 같다.

- 주문자의 성과 이름을 결합한 완전한 이름
- 주문자거주지의 완전한 주소
- 주문한 상품이름, 수량, 가격
- 기타 주문자의 전화번호, 수송날자, 주문날자

이러한 정보들은 자료기지설계요구로부터 여러 표에 나뉘여져 있다. 이 표들의 구조와 관계를 그림 6-3에 보여주었다.

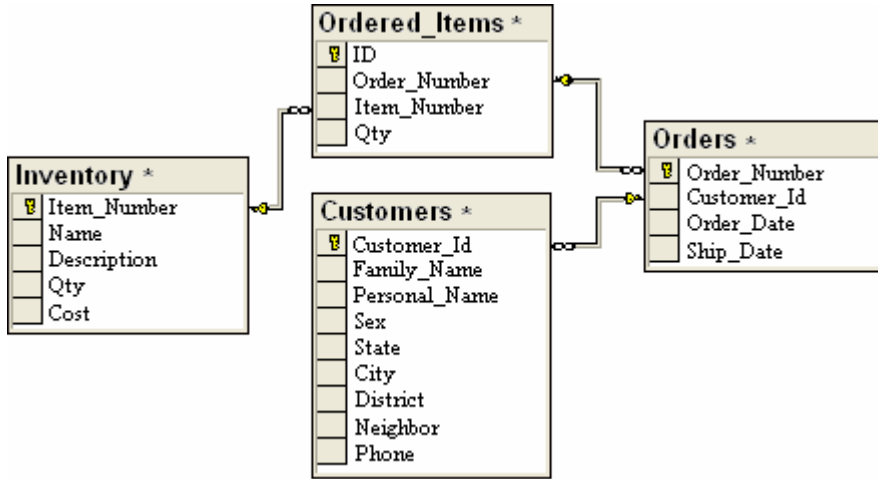



그림 6-3. 주문자정보를 포함하고있는 표들

봉사기측 코드:

필요한 주문자정보를 얻어내는 가장 좋은 방법은 GET_CUSTOMER_LIST라는 SQL저장수속을 정의하고 자료를 얻기 위하여 그것을 호출하는것이다.

목록 6-18. 주문자정보를 얻기 위한 저장수속

```
CREATE PROCEDURE GET_CUSTOMER_LIST AS
SELECT c.Family_Name + c.Personal_Name As Name,
   c.State+' '+c.City+' '+c.District+' '+c.Dong+' '+c.Neighbor As Address,
   i.Name As Goods, oi.Qty, i.Cost * 1.6 * oi.Qty AS Price,
   c.Phone, o.Order_Date, o.Ship_Date
FROM CUSTOMERS c, Orders o, Ordered_Items oi, Inventory i
WHERE o.Order_number = oi.Order_Number AND
     c.Customer_ID = o.Customer_ID AND
     i.Item_Number = oi.Item_Number;
```

표 6-4은 질문으로부터 얻어낸 주문자정보를 보여준다.

표 6-4. 주문자정보

Name	Address	Goods	Qty	Price	Phone	Order_Date	Ship_Date
김성철	평양시 평양 중구역 교구동 32	평양맥주	3	0.82	352-6532	2007-04-10	2007-04-12
김성철	평양시 평양 중구역 교구동 32	랭천 사이다	1	0.20	352-6532	2007-04-10	2007-04-12
김성철	평양시 평양 중구역 교구동 32	모란과자	2	0.99	352-6532	2007-04-10	2007-04-12
김은희	평양시 평양 보통강구역 대보동 17	박하사탕	6	2.40	453-5521	2007-04-05	2007-04-07
김은희	평양시 평양 보통강구역 대보동 17	고려 인삼술	2	6.24	453-5521	2007-04-05	2007-04-07
김은희	평양시 평양 보통강구역 대보동 17	강계인풍술	1	1.56	453-5521	2007-04-05	2007-04-07
김은희	평양시 평양 보통강구역 대보동 17	대동강 맥주	4	1.66	453-5521	2007-04-05	2007-04-07
김 철	함경남도 함흥 사포동 23	평양맥주	4	1.09		2007-04-09	2007-04-11
김 철	함경남도 함흥 사포동 23	살구씨향 과자	2	1.50		2007-04-09	2007-04-11
강권혁	평양시 평양 중구역 류성동 23	평양술	5	14.96	321-8710	2007-04-09	2007-04-11
강권혁	평양시 평양 중구역 류성동 23	백두산 들쭉술	2	6.56	321-8710	2007-04-09	2007-04-11

자료를 CachedRowSet로 얻기 위해서는 CachedRowSet객체를 만들고 저장수속을 실행하는데 리용한다. 목록 6-19에서 보여준 실례에서는 jdbc:odbc구동프로그람을 리용하여 적재한 Sun의 RowSet.jar파일로부터 CachedRowSet실현을 리용한다. 여기서 알수 있는바와 같이 코드는 CachedRowSet를 현시하는것이 아니라 serializeRows()메쏘드로 전체 CachedRowSet bean을 파일로 직렬화한다.

목록 6-19. CachedRowSet에서의 SQL질문실행

```
package JavaDB_Bible.ch06.sec02;

import java.io.*;
import java.sql.*;
import javax.sql.*;
import com.sun.rowset.*;

public class CachedRowSetSerializer {
    String fName = "ContactRowSet.ser";

    public static void main(String[] args) {
        CachedRowSetSerializer crs = new CachedRowSetSerializer();
        crs.serializeRows();
    }

    public CachedRowSetSerializer() {
    }

    public void serializeRows() {
        String url = "jdbc:odbc:CUSTOMERS";
        String login = "sa";
        String password = "dba";
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            CachedRowSet rowSet = new CachedRowSet();

            // 자료원천의 URL, 가입이름과 통과어를 설정
            rowSet.setUrl(url);
            rowSet.setUsername(login);
            rowSet.setPassword(password);

            // RowSet의 흘리기 및 갱신가능화
            rowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
            rowSet.setConcurrency(ResultSet.CONCUR_UPDATABLE);

            // SQL지령을 설정
            rowSet.setCommand("GET_CUSTOMER_LIST;");

            // 지령을 실행
            rowSet.execute();
```

```
            FileOutputStream fOut = new FileOutputStream(fName);
            ObjectOutput out = new ObjectOutputStream(fOut);
            out.writeObject(rowSet);
            out.flush();
            out.close();

            // RowSet의 닫기
            rowSet.close();
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

의뢰기측 코드:

의뢰기측코드는 더 간단하다. 목록 6-20에서 보여주는것처럼 직렬화된 CachedRowSet bean이 비직렬화(deserialize)되고 RowSet들은 조종탁에 출력된다. 실천적인 응용프로그람들은 아마도 간단한 GUI를 구동하는 구성요소로서 CachedRowSet bean을 리용할것이다.

목록 6-20. CachedRowSet의 리용

```
package JavaDB_Bible.ch06.sec02;

import java.io.*;
import sun.jdbc.rowset.*;

public class CachedRowSetDeserializer{
   public static void main(String[] args){
      CachedRowSetDeserializer crd = new CachedRowSetDeserializer();
      crd.deserializeRows(args[0]);
   }

   public CachedRowSetDeserializer(){
   }

   public void deserializeRows(String fName){
      try {
         FileInputStream fIn = new FileInputStream(fName);
         ObjectInputStream in = new ObjectInputStream(fIn);
         CachedRowSet rowSet = (CachedRowSet)in.readObject();
         while(rowSet.next()){
```

```
            for(int j=1;j<=rowSet.getMetaData().getColuranCount();j++){
               System.out.print( rowSet.getObject(j) +"\t");
            }
            System.out.println();
         }
         rowSet.close();
      }catch(Exception e){
         System.err.println(e.getMessage());
      }
   }
}
```

6.2.3. RowSet로부터 XML의 생성

RowSet객체의 설계자들은 RowSet가 XML프로그람에서 매우 쓸모있는 잠재력을 가질것이라는것을 예견하였다. WebRowSet는 RowSet를 XML형식으로 직렬화하거나 비직렬화하도록 설계된 CachedRowSet의 확장판이다. 이 클라스에는 RowSet를 XML형식으로 읽는데 리용하는 XmlReader객체와 RowSet를 XML형식으로 쓰는데 리용되는 XmlWriter객체가 저장되여있다.

주문자체계에서 주문자들에게 XML에 기초한 지불청구서를 보내는 경우를 생각해보자. 청구서에 기입할 자료는 목록 6-21에 보여준 저장수속 BILLING_DATA로부터 얻을수 있다. 어떤 특정한 주문자에 대한 지불청구자료를 얻으려면 그 주문자의 ID를 입력파라메터로 주어야 한다.

목록 6-21. 주문자체계에서 계산서를 얻기 위한 저장수속

```
CREATE PROCEDURE BILLING_DATA
@Id INT AS
SELECT o.Order_Number, c.Family_Name + c.Personal_Name As Name,
   c.State+' '+c.City+' '+c.District+' '+c.Dong+' '+c.Neighbor As Address,
   i.Name As Goods, oi.Qty, i.Cost * 1.6 * oi.Qty AS Price,
   c.Phone, o.Order_Date, o.Ship_Date
FROM CUSTOMERS c, Orders o, Ordered_Items oi, Inventory i
WHERE o.Order_Number = oi.Order_Number AND
     c.Customer_ID = o.Customer_ID AND
     i.Item_Number = oi.Item_Number AND c.Customer_Id = @Id;
```

목록 6-22는 Customer_Id가 101인 주문자에 대한 지불청구자료를 얻기 위하여 저장수속을 리용하는 실례를 보여준다. 알수 있는바와 같이 XML은 customer.xml이라는 파일로 작성된다.

목록 6-22. WebRowSet로 XML작성

```
package JavaDB_Bible.ch06.sec02;

import java.io.*;
import java.sql.*;
import javax.sql.*;
import sun.jdbc.rowset.*;

public class WebRowSetExample{
   public static void main(String[] argv){
      String url = "jdbc:odbc:CUSTOMERS";
      String login = "sa";
      String password = "dba";

      try{
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         WebRowSet rowSet = new WebRowSet();

         // 자료원천의 URL, 가입이름과 통과어 설정
         rowSet.setUrl( url );
         rowSet.setUsername( login );
         rowSet.setPassword( password );

         // sql지령 설정
         rowSet.setCommand("BILLING_DATA 101;");

         // 지령의 실행
         rowSet.execute();

         // RowSet를 XML로 출력
         FileWriter xmlFileWriter = new FileWriter("customer.xml");
         rowSet.writeXml(xmlFileWriter);

         // RowSet의 닫기
         rowSet.close();
      }catch(Exception e) {
         System.err.println(e.getMessage());
      }
   }
}
```

목록 6-23에서는 WebRowSet에 의하여 생성된 XML형식을 보여준다. WebRowSet는 XML형식으로 직렬화되기때문에 여기에는 행자료뿐만아니라 RowSet를 그대로 재구축하는데 필요한 모든 련관된 메타자료까지 다 포함되여있다.

목록 6-23. WebRowSet에 의하여 생성된 XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RowSet PUBLIC '-//Sun Microsystems, Inc.//DTD RowSet//EN'
'http://java.sun.com/j2ee/dtds/RowSet.dtd'>

<RowSet>
  <properties>
    <command>BILLING_DATA 101;</command>
    <concurrency>1008</concurrency>
    <datasource>
      <null/>
    </datasource>
    <escape-processing>true</escape-processing>
    <fetch-direction>0</fetch-direction>
    <fetch-size>0</fetch-size>
    <isolation-level>2</isolation-level>
    <key-columns></key-columns>
    <map></map>
    <max-field-size>0</max-field-size>
    <max-rows>0</max-rows>
    <query-timeout>0</query-timeout>
    <read-only>true</read-only>
    <rowset-type>1004</rowset-type>
    <show-deleted>false</show-deleted>
    <table-name>
      <null/>
    </table-name>
    <url>jdbc:odbc:CUSTOMERS</url>
  </properties>
  <metadata>
    <column-count>8</column-count>
    <column-definition>
      <column-index>1</column-index>
      <auto-increment>false</auto-increment>
      <case-sensitive>false</case-sensitive>
      <currency>false</currency>
      <nullable>0</nullable>
      <signed>false</signed>
```

```
    <searchable>true</searchable>
    <column-display-size>23</column-display-size>
    <column-label>family_name</column-label>
    <column-name>family_name</column-name>
    <schema-name></schema-name>
    <column-precision>23</column-precision>
    <column-scale>3</column-scale>
    <table-name></table-name>
    <catalog-name></catalog-name>
    <column-type>93</column-type>
    <column-type-name>datetime</column-type-name>
  </column-definition>

  <-- 나머지 7개의 렬정의요소들은 생략 -->

  </metadata>
  <data>
    <row>
      <col>김은희</col>
      <col>평양시 평양 보통강구역 대보동 17</col>
      <col>고려인삼술</col>
      <col>2</col>
      <col>6.24</col>
      <col>453-5521</col>
      <col>2007-04-05 00:00:00.0</col>
      <col>2007-04-07 00:00:00.0</col>
    </row>
  </data>
</RowSet>
```

WebRowSet객체에 의하여 만들어진 XML파일은 세가지 부분으로 나뉘여져있다.

- <properties> - 이 부분에는 WebRowSet Bean과 관련된 모든 속성자료가 포함된다.
- <metadata> - 이 부분에는 매 렬들을 설명하는 메타자료요소가 포함된다.
- <data> - 이 부분에는 실제자료가 포함된다.

8개의 렬들에 대한 자료가 모두 비슷하므로 목록 6-23에서는 하나의 렬정의요소만을 보여주었다.

이 XML출력이 그대로는 응용프로그람의 요구를 만족시키지 못할수 있다. 이 경우에 대처하기 위한 세가지 방도가 있다.

- WebRowSet를 리용하여 만든 XML로부터 목적하는 XML을 생성하기 위한 XSL변환의 적용
- 목적하는 XML을 직접 생성하기 위한 전용 XmlWriter의 작성
- CachedRowSet로부터 직접 XML문서를 생성

우리의 실례에서는 0020CachedRowSet로부터 직접 XML문서를 생성하는 방법을 리용한다. 그 리유는 전체 Bean을 직렬화하기 위해 설계된 WebRowSet가 응용프로그람에서 요구하는것보다 훨씬 더 많은 자료를 포함하고있기때문이다.

XSL변환을 리용하는것은 대단히 복잡한 방법이다. 류사하게 전용 XmlWriter를 작성하는것도 응용프로그람에 요구되는 XML을 작성하는것보다 훨씬 더 복잡하다. 즉 Bean을 직렬화하는데 필요한 모든 자료를 다 써내도록 XmlWriter를 설계할뿐아니라 대응하는 XmlReader로 그것을 지원해야 한다.

XML은 파일로서 의뢰기에 전송되도록 되여있으므로 목록 6-24에서 보는바와 같이 문자렬들을 OutputStream에 써넣는것에 의해 생성된다. 만일 XML에 대한 추가적인 처리를 더 하려면 Xerces Document객체를 리용할수 있으며 그것을 DOM으로 구축할수 있다.

목록 6-24. CachedRowSet를 리용한 XML의 생성

```
package JavaDB_Bible.ch06.sec02;

import java.io.*;
import sun.jdbc.rowset.*;

public class CachedRowSetToXML{
   public static void main(String[] argv){
      String url = "jdbc:odbc:CUSTOMERS";
      String login = "sa";
      String password = "dba";
      String fileName = "customer.xml";
      try{
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         CachedRowSet rowSet = new CachedRowSet();
         PrintWriter out = new PrintWriter(new FileOutputStream(fileName));

         rowSet.setUrl( url );
         rowSet.setUsemame( login );
         rowSet.setPassword( password );

         rowSet.setCommand("BILLING_DATA 101;");
```

```
        rowSet.execute();
        out.println("<?xml version=\"l.0\" encoding=\"UTF-8\"?>");
        out.println("  <customers>");

        while(rowSet.next()){
          out.println("    <customer>");
           for(int j=1; j<=rowSet.getMetaData().getColumnCount();j++) {
            out.print("     <"+
                    rowSet.getMetaData().getColumnLabel(j)+">");
            out.print(rowSet.getObject(j));

out.println("</"+rowSet.getMetaData().getColumnLabel(j)+">");
          }
          out.println("    </customer>");
        }
        out.println("  </customers>");

        rowSet.close();
        out.close();
      }catch(Exception e) {
        e.printStackTrace();
      }
   }
}
```

이 실례에서 만든 XML을 목록 6-25에서 보여주었다.

목록 6-25. XML주문자요소들

```
<?xml version= "l.0" encoding="UTF~8"?>
<customers>
   <customer>
      <Name>김은희</Name>
      <Address>평양시 평양 보통강구역 대보동 17</Address>
      <Goods>고려인삼술</Goods>
      <Qty>2</Qty>
      <Price>6.24</Price>
      <Phone>453-5521</Phone>
      <Order_Date>2007-04-05 00:00:00.0</Order_Date>
      <Ship_Date>2007-04-07 00:00:00.0</Ship_Date>
   </customer>
   <customer>
      <Name>김은희</Name>
```

```
    <Address>평양시 평양 보통강구역 대보동 17</Address>
    <Goods>강계인풍술</Goods>
    <Qty>1</Qty>
    <Price>1.56</Price>
    <Phone>453-5521</Phone>
    <Order_Date>2007-04-05 00:00:00.0</Order_Date>
    <Ship_Date>2007-04-07 00:00:00.0</Ship_Date>
  </customer>
  <customer>
    <Name>김은희</Name>
    <Address>평양시 평양 보통강구역 대보동 17</Address>
    <Goods>대동강맥주</Goods>
    <Qty>4</Qty>
    <Price>1.66</Price>
    <Phone>453-5521</Phone>
    <Order_Date>2007-04-05 00:00:00.0</Order_Date>
    <Ship_Date>2007-04-07 00:00:00.0</Ship_Date>
  </customer>
  <customer>
    <Name>김은희</Name>
    <Address>평양시 평양 보통강구역 대보동 17</Address>
    <Goods>박하사탕</Goods>
    <Qty>6</Qty>
    <Price>2.40</Price>
    <Phone>453-5521</Phone>
    <Order_Date>2007-04-05 00:00:00.0</Order_Date>
    <Ship_Date>2007-04-07 00:00:00.0</Ship_Date>
  </customer>
</customers>
```

제3절. SQL을 리용한 XML문서의 접근

이 절에서는 JDBC접근가능한 XML자료기지관리체계구축과 SQL을 리용한 XML문서의 질문작성에 대하여 보기로 한다.

6.3.1. SQL로 XML문서에 접근하는 근거

XML의 일차적인 리용은 응용프로그람들사이에 가동기반독립의 자료운반이지만 다른 하나의 중요한 용도는 국부적인 자료저장이다. 자료저장기로 리용되는 일반적인 실례들은 다음과 같다.

- XML을 속성파일이나 INI파일대신 리용한다.
- 본문자료기지에서 반점으로 구분된 CSV파일*대신으로 리용한다.
- 주식시세 혹은 새소식표제의 송달을 위한 내리적재가능한 작은 자료기지로 리용한다.

어떤 경우 자료를 저장하고있는 XML문서는 그 자체가 자료기지일수 있다. 레를 들어 PDA에서 주문자목록은 XML문서들로 보관할수 있다.

XML파일에서 자료는 두가지 다른 마디형으로 저장되기때문에 XML파일을 리용하여 자료기지를 구축하는데는 두가지 방법이 있다.

- 매 레코드를 속성안에 마당자료가 있는 요소로 저장
- 매 레코드를 자식요소들속에 마당자료가 있는 요소로 저장

속성을 리용하는 방법의 우점은 속성이름이 한번만 나타나기때문에 XML파일이 보다 짧아지는 우점이 있다. 만일 자료를 자식요소로 저장한다면 이름이 두번 즉 요소를 열 때와 닫을 때 각각 나타난다.

아래에서는 속성에 기초한 자료저장방법을 보여주고있다.

```
<FONT FACE="Arial" SIZE="2" COLOR="#000000"/>
```

매 자료항목에 대하여 자식요소들을 리용하는 방법은 아래에 보여준것처럼 장황하지만 보다 구조화되여있다.

---

* CSV(Comma Separated Value)형식은 서로 다른 응용프로그람들사이에 자료를 교환할수 있도록 자료의 매 항목을 반점으로 구분한 본문파일이다. 다시말하여 CSV파일은 자료교환을 위한 표준파일형식이다.

```
<?xml version="1.0"?>
<CUSTOMERS>
  <CUSTOMER CUSTOMER_ID="100">
    <FAMILY_NAME>김</FAMILY_NAME>
    <PERSONAL_NAME>성철</PERSONAL_NAME>
    <SEX>남</SEX>
    <STATE>평양시</STATE>
    <CITY>평양</CITY>
    <DISTRICT>중구역</DISTRICT>
    <DONG>교구동</DONG>
    <NEIGHBOR>32</NEIGHBOR>
    <PHONE>326-6532</PHONE>
  </CUSTOMER>
</CUSTOMERS>
```

이 절에서 설명하게 되는 JDBC구동프로그람은 전용자료형 ATTRIBUTE를 정의하여 자료의 삽입을 속성으로 지원해준다. 다른 자료형들은 모두 자식요소로 삽입된다. 실천적으로는 XML문서의 모든 자료가 문자렬로 표현되기때문에 한가지 다른 자료형 VARCHAR만이 있을수 있다. JDBC구동프로그람에 대한 구체적인 내용은 다음 소절에서 고찰한다.

6.3.2. JDBC접근가능한 XML 자료기지관리체계구축

JDBC프로그람작성대면부를 결합시키는 XML자료기지체계를 구축하기 위해서는 두가지 주되는 구성성분 즉 JDBC구동프로그람 클라스들과 SQL엔진이 요구된다.

실현기초클라스

JDBC API는 개발자가 하고 싶은것은 어느것이나 처리할수 있는 메쏘드들을 충분히 갖추고있다. 이 메쏘드들은 대면부클라스들의 모임에 명기되여있다. DriverManager와 함께 등록될수 있는 JDBC구동프로그람을 작성하려면 이 대면부의 메쏘드들이 실현되여야 한다. 이것은 실현기초클라스(implementation base class)들을 리용하여 수행된다.

실현기초클라스들은 대면부에서 정의된 모든 메쏘드들을 최소한의 형식으로 실현하고 있다. 이 메쏘드들은 호출될 때 단순히 례외를 던진다. JDBC구동프로그람 클라스들은 실현기초클라스들의 확장이며 일감을 수행하는데 필요한 메쏘드들을 재정의한다. 목록 6-26에서는 이 실현클라스들중 한 클라스의 일부분을 보여주었다.

목록 6-26. 전형적인 실현기초클라스

```
package JavaDB_Bible.ch06.sec03.JDBCImpl;

import java.sql.*;

public class JDBCStatementImpl implements java.sql.Statement {
  public JDBCStatementImpl(){
  }

  public void setFetchSize(int fetchSize) throws SQLException {
    throw new SQLException("not supported");
  }

  public int getFetchSize() throws SQLException {
    throw new SQLException("not supported");
  }
```

단순히 례외만을 던지는 수백개의 메쏘드들을 정확히 입력하는것이 시끄러우면 웨브싸이트로부터 실현클라스들을 내려싣기할수 있다. 혹은 응용프로그람에서 구동프로그람을 java.sql.DriverManager와 함께 등록할 필요가 없다면 간단히 클라스정의에서 extends문절을 없애면 된다.

SQL엔진도 제한적이지만 역시 확장할수 있다. SQL엔진은 SQL지령들의 기초적이며 일반적인 분석을 조종한다. 이 코드는 임의의 SQL프로그람에 다 적용할수 있다. 우리의 프로그람은 문서의 작성 및 갱신과 함께 일반적인 질문들을 처리할수 있게 하는 가능한 지령들의 부분모임만을 실현한다.

XML문서처리기들은 기초적인 SQL엔진클라스들을 확장하여 XML특정의 자료접근과 갱신능력을 제공한다. 만일 XML문서가 아니라 배렬과 함께 리용하는 자기식의 저장매체조종기를 만들려고 한다면 그것들을 대신 끼워넣을수 있다.

JDBC클라스들의 실현

JDBC의 내부작업은 앞서 배운 장들에서 얼마간 취급되였다. 아래에서는 클라스들이 서로 어떻게 작업하는가에 대하여 간단히 개괄한다.

① XMLDriver클라스

DriverManager의 역할은 JDBC구동프로그람들을 관리하기 위한 기초적인 봉사들을 제공하는것이다. 구동프로그람들은 Class.forName()을 리용하여 초기화시 혹은 요청시

에 적재될수 있다. 모든 구동프로그람들은 구동프로그람의 구체례를 창조하고 그것을 DriverManager와 함께 등록하는 정적인 초기화프로그람을 포함하고있다. 목록 6-27은 XMLDriver클라스가 얼마나 간단한가를 보여준다.

목록 6-27. XMLDriver클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.sql.*;
import java.util.Properties;
import JavaDB_Bible.ch06.sec03.JDBCImpl.JDBCDriverImpl;

public class XMLDriver extends JDBCDriverImpl {
    protected XMLConnection con;

    static {
        try {
            java.sql.DriverManager.registerDriver(new XMLDriver());
        } catch (SQLException e) {
            System.err.println(e);
        }
    }

    public XMLDriver() {
    }

    public boolean acceptsURL(String url) throws SQLException {
        return url.endsWith(".xml");
    }

    public XMLConnection connect(String url) throws SQLException {
        con = new XMLConnection(url);
        return con;
    }
}
```

DriverManager는 JDBC구동프로그람들을 적재하는 기능과 함께 지적된 URL에 해당한 구동프로그람의 위치를 찾아서(등록된 구동프로그람들의 acceptURL(String url) 메쏘드를 조사한다.) 그 구동프로그람에 고유한 접속을 돌려준다.

DriverManager는 또한 자료기지에 대한 java.sql.Connection을 얻어야 한다.

DriverManager는 구동프로그람의 connect()메쏘드를 호출하여 자료기지에 대한 URL을 넘겨준다. 이때 구동프로그람은 Connection객체를 창조하고 그것을 DriverManager에 돌려준다.

② XMLConnection클라스

java.sql.Connection은 특정한 자료기지 혹은 이 경우 특정한 XML문서와의 대화접속을 나타낸다. XML문서는 Connection객체에 넘겨진 URL에 의해 정해진다. 이때 Connection객체는 URL규약에 따라 다음 방법들중의 하나로 그 URL에 대한 접속을 시도한다.

- 만일 URL규약이 그 문서가 파일이라고 지적하면 Connection은 그 파일의 열기를 시도한다.
- 만일 URL규약이 HTTP접속이라고 지적하면 Connection은 그 URL에 접속하여 XML문서를 열기 위해 시도한다.

일단 이미 존재하는 파일이라든가 HTTP자료원천에 대한 접속이 확립되면 XML문서는 DOM문서로 분석된다. URL에 파일이 존재하지 않는 경우에는 새로운 DOM문서가 창조된다. 목록 6-28에서는 XMLConnection클라스를 보여주고있다.

목록 6-28. XMLConnection클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.io.*;
import java.net.*;
import java.sql.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.InputSource;
import org.apache.xerces.parsers.DOMParser;
import org.apache.xerces.dom.DocumentImpl;
import JavaDB_Bible.ch06.sec03.JDBCImpl.JDBCConnectionImpl;

public class XMLConnection extends JDBCConnectionImpl {
  private URL url;
  private Document xmlDoc;

  public XMLConnection() throws SQLException {
  }
```

```
public XMLConnection(InputSource xml) throws SQLException {
  try {
    DOMParser p = new DOMParser();
    p.parse(xml);
    xmlDoc = p.getDocument();
  } catch (Exception e) {
    System.err.println(e);
  }
}

public XMLConnection(URL url) throws SQLException {
  this.url = url;
  xmlDoc = setXmlDoc(url);
}

public XMLConnection(String UrlString) throws SQLException {
  try {
    xmlDoc = setXmlDoc(new URL(UrlString));
  } catch (Exception e) {
    System.err.println(e);
  }
}

private Document setXmlDoc(URL url) {
  Document xmlDoc = null;

  if (url.getProtocol().equalsIgnoreCase("file")) {
    File f = new File(url.getFile());
    if (!f.exists()) {
      String rootTag = f.getName();
      rootTag = rootTag.substring(rootTag.lastIndexOf("/") + 1,
                   rootTag.indexOf("."));
      return createXmlDoc(rootTag);
    }
  }

  try {
    InputStream s = url.openStream();
    if (s != null) {
      DOMParser p = new DOMParser();
      p.parse(new InputSource(s));
```

```
        xmlDoc = p.getDocument();
      }
    } catch (Exception e) {
      System.err.println(".." + e);
    }
    return xmlDoc;
  }

  private Document createXmlDoc(String rootTag) {
    xmlDoc = new DocumentImpl();
    Element root = (Element) xmlDoc.createElement(rootTag);
    xmlDoc.appendChild(root);
    return xmlDoc;
  }

  public Statement createStatement() {
    return new XMLStatement(xmlDoc);
  }

  public XMLStatement createStatement(URL url) {
    return new XMLStatement(xmlDoc);
  }
}
```

Connection객체는 자료원천에 대한 접속외에 자기의 createStatement()메쏘드가 호출될 때 Statement객체를 돌려주는 책임도 진다. createStatement()메쏘드는 새로운 Statement객체를 창조하고 그것을 Connection에 포함된 DOM문서에 넘겨준다.

③ XMLStatement클라스

java.sql.Statement객체는 execute()와 executeQuery()메쏘드들에 대한 제일 웃준위 지령해석기로 작용한다. 여기서는 보다 간단한 CREATE지령과 INSERT지령들이 국부적으로 처리되고 질문들은 XMLQuery객체에 의해 처리된다.

Statement객체의 기본적인 메쏘드들은 다음과 같다.

- public ResultSet executeQuery(String sqlQuery) – executeQuery()메쏘드는 새로운 XMLQuery객체를 창조하고 SQL질문을 처리하는데 리용된다. XMLQuery객체는 목록 6-34에서 설명하였다.
- public int executeUpdate(String sqlString) – 이 메쏘드는 새로운 XMLCommand객체를 창조하고 그것을 createTable()이나

insert()메쏘드에 보낸다.

- private boolean createTable(XMLCommand sql) -이 메쏘드는 XMLCommand의 splitColumns()메쏘드를 리용하여 렬목록을 렬벡토르로 돌려준다. 표는 벡토르 columnNameVector와 columnType Vector를 리용하여 정의된다.
- private boolean insert(XMLCommand sql) - 이 메쏘드는 columnNameVector와 columnTypeVector를 리용하여 매 자료마당에 대한 XML요소를 생성하고 그것들을 삽입된 행을 나타내는 행요소안에 밀어넣는다.

자료가 속성으로 추가된다는것을 지적하는 전용자료형 ATTRIBUTE에 대한 참조에 류의하기 바란다. XMLStatement객체는 두가지 방법중의 하나로 SQL INSERT지령을 처리한다. 만일 자료형이 ATTRIBUTE이면 자료문자렬은 XML요소에 속성으로 삽입된다. 그렇지 않으면 그것은 요소로 추가된다. 목록 6-29는 XMLStatement객체에 대한 코드를 보여준다.

목록 6-29. XMLStatement클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.util.Vector;
import java.sql.*;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.xml.sax.InputSource;
import JavaDB_Bible.ch06.sec03.JDBCImpl.JDBCStatementImpl;

public class XMLStatement extends JDBCStatementImpl {
   private InputSource xml;
   private Document xmlDoc;
   private Vector columnNameVector = new Vector();
   private Vector columnTypeVector = new Vector();

   public XMLStatement() {
   }

   public XMLStatement(InputSource xml) {
```

```
        this.xml = xml;
    }

    public XMLStatement(Document xmlDoc) {
        this.xmlDoc = xmlDoc;
    }

    public ResultSet executeQuery(String sqlQuery) throws SQLException {
        XMLQuery query = new XMLQuery(sqlQuery);
        return query.processDoc(xmlDoc);
    }

    public int executeUpdate(String sqlString) {
        XMLCommand sql = new XMLCommand(sqlString);
        if (sql.cmd.equals("CREATE")) createTable(sql);
        if (sql.cmd.equals("INSERT")) insert(sql);
        return 0;
    }

    private boolean createTable(XMLCommand sql) {
        Vector columnVector = sql.splitColumns(sql.columns);
        for (int i = 0; i < columnVector.size(); i++) {
            String columnDef = ((String) columnVector.elementAt(i)).trim();
            int space = columnDef.indexOf(" ");
            if (space >= 0) {
                String colName = columnDef.substring(0, space);
                String colType = columnDef.substring(space + 1);
                columnNameVector.addElement(colName);
                columnTypeVector.addElement(colType);
            }
        }
        return true;
    }

    private void initColumnData(XMLCommand sql) {
        NodeList records = xmlDoc.getElementsByTagName(sql.tableName);
        Element record = (Element) records.item(0);
        NamedNodeMap attribs = record.getAttributes();

        for (int i = 0; i < attribs.getLength(); i++) {
            Node n = attribs.item(i);
            if (n.getNodeType() == Node.ATTRIBUTE_NODE) {
```

```
            columnNameVector.addElement(n.getNodeName());
            columnTypeVector.addElement("ATTRIBUTE");
        }
    }

    NodeList fields = record.getChildNodes();
    for (int i = 0; i < fields.getLength(); i++) {
        Node n = fields.item(i);
        if (n.getNodeType() == Node.ELEMENT_NODE) {
            Element field = (Element) n;
            columnNameVector.addElement(field.getTagName());
            columnTypeVector.addElement("VARCHAR");
        }
    }
}

private boolean insert(XMLCommand sql) {
    if (columnNameVector.isEmpty()) initColumnData(sql);
    Vector data = sql.splitValues(sql.values);

    try {
        Element root = xmlDoc.getDocumentElement();
        Element row = (Element) xmlDoc.createElement(sql.tableName);
        root.appendChild(row);
        for (int i = 0; i < data.size(); i++) {
            String cName = (String) columnNameVector.elementAt(i);
            String cType = (String) columnTypeVector.elementAt(i);
            String cData = (String) data.elementAt(i);
            if (cData.startsWith("'") && cData.endsWith("'")) {
                if (cData.length() > 1) {
                    cData = cData.substring(1, cData.length() - 1);
                }
            }

            if (cType.equals("ATTRIBUTE")) {
                row.setAttribute(cName, cData);
            } else {
                Element column = xmlDoc.createElement(cName);
                row.appendChild(column);
                column.appendChild(xmlDoc.createTextNode(cData));
            }
        }
```

```
        } catch (Exception e) {
            System.err.println("Insert error: " + e);
        }
        return true;
    }

    public Document getXmlDocument() {
        return xmlDoc;
    }
}
```

XMLQuery객체는 java.sql.ResultSet를 실현하는 XMLResultSet를 돌려준다. 이 ResultSet는 질문이 돌려주는 자료를 포함하고있는 용기이다. 이 프로그람에서는 XML을 취급하므로 ResultSet는 지적된 요소와 자식요소들을 포함하는 DOM문서로 유지된다.

XMLResultSet

XMLResultSet의 가장 중요한 메쏘드들은 next()와 getString()이다. next()메쏘드는 ResultSet를 구성하고있는 마디들을 차례차례 반복하기 위하여 NodeList행들을 리용한다. next()메쏘드는 처음 호출되면 ResultSet문서로부터 NodeList를 초기화한다. 다음 int rowIndex에 현재 행을 가리키는 유표를 넣는다.

목록 6-30에서 보여준 getString()메쏘드는 렬이름변형만을 리용하여 실현되였다. 그 리유는 다음과 같다.

- XMLResultSet가 속성으로 저장된 자료 혹은 요소로 저장된 자료를 지원하므로 위치는 무의미하다.
- XML처리기는 자식마디의 순서를 우연적으로 정한다. 따라서 위치가 상관없다.

실례에서 getString()이 속성마디를 먼저 찾고 다음에 그에 알맞는 요소를 어떻게 찾는가에 주의하기 바란다. 속성을 얻는것이 보통 더 빠르다.

목록 6-30. XMLResultSet클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.io.*;
import java.sql.*;
import org.w3c.dom.Document;
```

```
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.apache.xerces.dom.DocumentImpl;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import JavaDB_Bible.ch06.sec03.JDBCImpl.JDBCResultSetImpl;

public class XMLResultSet extends JDBCResultSetImpl {
    private int rowIndex = -1;
    private int rowCount = 0;

    public Document xmlDoc;
    private Element root = null;
    private Element currentRow = null;
    private NodeList rows = null;

    public XMLResultSet() throws SQLException {
        xmlDoc = new DocumentImpl();
        root = (Element) xmlDoc.createElement("RESULTSET");
        xmlDoc.appendChild(root);
    }

    private void initialise() {
        if (rows == null) {
            root = xmlDoc.getDocumentElement();
            rows = root.getChildNodes();
            rowCount = rows.getLength();
        }
    }

    public boolean next() {
        if (rows == null) initialise();

        if (++rowIndex == rowCount) {
            return false;
        } else {
            currentRow = (Element) rows.item(rowIndex);
            return true;
        }
    }
```

```
public boolean previous() {
    if (rows == null) {
        initialise();
        rowIndex = rowCount;
    }

    if (--rowIndex < 0) {
        return false;
    } else {
        currentRow = (Element) rows.item(rowIndex);
        return true;
    }
}

public boolean absolute(int row) throws SQLException {
    if (row == 0)throw new SQLException("invalid row number");
    boolean onValidRow = true;

    if (rows == null) initialise();

    if (row > 0) rowIndex = row - 1;
    else if (row < 0) {
        rowIndex = rowCount + row;
    }

    if (rowIndex < -1) {
        rowIndex = -1;
        onValidRow = false;
    }

    if (row > rowCount) {
        rowIndex = rowCount;
        onValidRow = false;
    }

    currentRow = (Element) rows.item(rowIndex);
    return onValidRow;
}

public boolean relative(int row) throws SQLException {
    boolean onValidRow = true;
    if (rows == null) initialise();
```

```
        return absolute(row + rowIndex + 1);
    }

    public void beforeFirst() throws SQLException {
        if (rows == null) initialise();
        rowIndex = -1;
    }

    public void afterLast() throws SQLException {
        if (rows == null) initialise();
        rowIndex = rowCount;
    }

    public boolean first() throws SQLException {
        if (rows == null) initialise();
        return absolute(1);
    }

    public boolean last() throws SQLException {
        if (rows == null) initialise();
        return absolute( -1);
    }

    // 먼저 렬이름과 맞는 속성이 있는가를 찾고 다음 자식요소가 있는가를 찾는다
    public String getString(String columnName) throws SQLException {
        if (currentRow == null)
            throw (new SQLException("Invalid row: " + currentRow));
        String value = currentRow.getAttribute(columnName);

        if (value.length() > 0) {
            return value;
        } else {
            NodeList cols = currentRow.getElementsByTagName(columnName);
            if (cols.getLength() > 0) {
                Node column = cols.item(0);
                NodeList children = column.getChildNodes();
                for (int i = 0; i < children.getLength(); i++) {
                    if (children.item(i).getNodeType() == Node.TEXT_NODE)
{
                        return (String) children.item(i).getNodeValue();
                    }
                }
            }
```

```
        }
        return null;
    }

    public ResultSetMetaData getMetaData() throws SQLException {
        return new XMLResultSetMetaData(this);

    }

    // 결과문서를 직렬화하기 위한 편의메쏘드
    public void serializeAsFile(String fileName) {
        try {
            OutputFormat fmt = new OutputFormat("xml", null, true);
            XMLSerializer serializer =
                    new XMLSerializer(new FileWriter(fileName), fmt);
            serializer.asDOMSerializer().serialize(xmlDoc);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

① 흘리기가능한 ResultSet의 실현

목록 6-30에서 알수 있는것처럼 흘리기가능한 ResultSet를 만드는데 필요한 메쏘드들을 실현하는것은 상대적으로 쉽다. XMLResultSet를 흘리기가능한 ResultSet로 만들기 위해서는 다음의 메쏘드들을 추가해야 한다.

- previous() - 한번에 한 행씩 유표를 뒤로 움직인다.
- first() - 유표를 맨 처음 행으로 이동
- last() - 유표를 맨 마지막 행으로 이동
- beforeFirst() - 유표를 맨 첫 행의 바로 앞으로 이동
- afterLast() - 유표를 맨 마지막 행의 바로 뒤로 이동
- absolute(int rowNumber) - 유표를 특정한 행으로 이동
- relative(int rowNumber) - 유표를 특정한 행수만큼 이동

목록 6-31에서는 absolute()메쏘드에 요구하는 행을 계산하여 넣어줌으로써 다른 모든 유표이동메쏘드들을 만들었다. 실례로 first()메쏘드는 absolute()메쏘드에 1을 인수로 주어 호출한다. relative()메쏘드도 행을 계산하여 absolute()메쏘드를 호출한다.

목록 6-31. 흘리기가능한 ResultSet메쏘드들

```
public boolean previous(){
   if(rows==null){
      initialise();
      rowIndex = rowCount;
   }
   if(--rowIndex < 0){
      return false;
   }else{
      currentRow = (Element)rows.item(rowIndex);
      return true;
   }
}

public boolean absolute(int row) throws SQLException {
   if(row == 0)throw new SQLException("invalid row number");
   boolean onValidRow = true;
   if(rows==null) {
      initialise();
   }

   if(row > 0) rowIndex = row-1;
   else if(row < 0){
      rowIndex = rowCount + row;
   }

   if(rowIndex<-1){
      rowIndex=-1;
      onValidRow = false;
   }
   if(row > rowCount){
      rowIndex = rowCount;
      onValidRow = false;
   }
   currentRow = (Element)rows.item{rowIndex);
   return onValidRow;
}

public boolean relative(int row) throws SQLException {
   boolean onValidRow = true;
   if(rows==null){
      initialise();
```

```
    }
    return absolute(row + rowIndex + 1);
}

public void beforeFirst() throws SQLException {
    if(rows==null){
        initialise();
    }
    rowIndex = -1;
}

public void afterLast() throws SQLException {
    if(rows==null){
        initialise();
    }
    rowIndex = rowCount;
}

public boolean first() throws SQLException {
    if{rows==null){
        initialise();
    }
    return absolute(1);
}

public boolean last() throws SQLException {
    if(rows==null){
        initialise();
    }
    return absolute(-1);
}
```

흘리기가능한 ResultSet를 보다 완성하려면 유표위치를 알려주는 보충적인 메쏘드들이 필요하다.

또한 TYPE_FORWARD_ONLY 혹은 TYPE_SCROLL_SENSITIVE와 같은 각이한 ResultSet형에 따르는 요구를 조종하기 위한 수법들도 요구된다.

② XMLResultSetMetaData클라스

XMLResultSet는 ResultSetMetaData대면부를 실현한 XMLResultSetMetaData에 의해 지원된다. XMLResultSetMetaData클라스는 ResultSet에 있는 렬수, 렬이름, 렬자료형 등의 편의정보를 제공한다. 그 코드를 목록 6-32에 보여주었다.

목록 6-32. XMLResultSetMetaData클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.sql.*;
import java.util.Vector;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import JavaDB_Bible.ch06.sec03.JDBCImpl.JDBCResultSetMetaDataImpl;

public class XMLResultSetMetaData extends JDBCResultSetMetaDataImpl {
  private XMLResultSet rs;
  private NodeList rows = null;
  private NodeList cols = null;
  private Vector columnNameVector = new Vector();

  public XMLResultSetMetaData(XMLResultSet rs) {
    this.rs = rs;
    Element root = rs.xmlDoc.getDocumentElement();
    rows = root.getChildNodes();
    Element currentRow = (Element) rows.item(0);
    NodeList children = currentRow.getChildNodes();
    for (int i = 0; i < children.getLength(); i++) {
      if (children.item(i).getNodeType() == Node.ELEMENT_NODE) {
        columnNameVector.addElement(((Element) children.item(i)).
                    getTagName());
      }
    }
  }

  public int getColumnCount() throws java.sql.SQLException {
    return columnNameVector.size();
  }

  public String getColumnLabel(int column) throws java.sql.SQLException
{
    return (String) columnNameVector.elementAt(column);
  }

  public String getColumnName(int column) throws java.sql.SQLException
{
```

```
    return (String) columnNameVector.elementAt(column);
  }

  public int getColumnType(int column) throws java.sql.SQLException {
    return java.sql.Types.VARCHAR;
  }

  public String getColumnTypeName(int column) throws java.sql.SQLException {
    return "VARCHAR";
  }

  public String getTableName(int column) throws java.sql.SQLException {
    return ((Element) rows.item(0)).getTagName();
  }
}
```

SQL엔진의 실현

SQL엔진은 SQL지령들을 다양한 구성요소들로 분석하는것과 함께 WHERE복합문절의 론리를 평가하는것과 같은 보다 세부적인 처리를 수행한다. 이 프로그람에서는 SQL-92 지령모임의 작은 부분모임만을 실현하지만 XML문서들을 작성하고 갱신하는것은 물론 일반질문들을 처리하는데 충분하다.

질문부분모임은 표 6-5에서 보여준 질문류형들을 처리하는것으로 제한된다.

표 6-5. 지원되는 질문연산자들

Function	Operator	Comment
동등성	=	String.equals()
비동등성	<>	
비교	LIKE	%통용문자들을 지원한다.
부정	NOT	부정연산자

SQL지령들을 처리하는데 리용되는 기초클라스는 XMLCommand이다. 이 클라스는 비록 이름에는 XML이 붙어있지만 XML에 국한된것이 아니다. XML에 특정한 기능은 이 클라스의 확장으로 부여된다. 이런식으로 클라스는 임의의 다른 SQL엔진의 기초로서 리용될수 있다.

① XMLCommand클라스

XMLCommand클라스의 기본 메쏘드들중의 하나는 지령을 그의 주요 구성요소문절들로 가르는데 리용되는 parseSQLCmd()이다. 구성요소문절들로는 지령 그자체, 표이름, 렬이름, WHERE문절을 들수 있다. splitFields()같은 추가적인 편의메쏘드들은 이 문절들을 더 처리하기 위하여 부분문절들의 벡토르들로 분할한다. 목록 6-33은 XMLCommand클라스를 보여준다.

목록 6-33. XMLCommand클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.sql.*;
import java.net.*;
import java.util.*;

public class XMLCommand {
   protected String SQLString;
   protected String cmd = null;
   protected String tableName = null;
   protected String columns = null;
   protected String values = null;
   protected String fields = null;
   protected String where = null;
   protected String orderBy = null;

   public XMLCommand() {
   }

   public XMLCommand(String SQLString) {
      this.SQLString = SQLString.toUpperCase().trim();
      parseSQLCmd(SQLString);
   }

   protected void parseSQLCmd(String SQLCmd) {
      cmd = SQLCmd.substring(0, SQLCmd.indexOf(" "));
      tableName = getTableName(SQLCmd);

      int tNameEnds = SQLCmd.indexOf(tableName) + tableName.length();
      int columnsEnd = SQLCmd.indexOf(" VALUES");
      int valuesIndex = SQLCmd.indexOf(" VALUES");
      int fromIndex = SQLCmd.indexOf(" FROM ");
      int whereIndex = SQLCmd.indexOf(" WHERE ");
      int orderIndex = SQLCmd.indexOf(" ORDER ");
      int orderByIndex = SQLCmd.indexOf(" BY ", orderIndex);
```

```
      if (whereIndex > -1) whereIndex += " VALUES".length();
      if (valuesIndex > -1) valuesIndex += " VALUES".length();

      if (cmd.equals("CREATE")) {
         columns = SQLCmd.substring(tNameEnds).trim();
      } else if (cmd.equals("INSERT")) {
         columns = SQLCmd.substring(tNameEnds, columnsEnd).trim();
         values = SQLCmd.substring(valuesIndex).trim();
      } else if (cmd.equals("SELECT")) {
         fields = SQLCmd.substring("SELECT".length(), fromIndex).trim();
         if (whereIndex > -1) {
            if (orderIndex > -1) {
               where = SQLCmd.substring(whereIndex, orderIndex);
            } else {
               where = SQLCmd.substring(whereIndex);
            }
            where = where.trim();
         }
         if (orderIndex > -1) {
            orderBy = SQLCmd.substring(orderByIndex).trim();
         }
      }
   }


   private String getTableName(String SQLCmd) {
      String tableName = null;
      if (SQLCmd.startsWith("SELECT")) {
         tableName = wordAfter(SQLCmd, "FROM");
      } else if (SQLCmd.startsWith("INSERT")) {
         tableName = wordAfter(SQLCmd, "INTO");
      } else if (SQLCmd.startsWith("UPDATE")) {
         tableName = wordAfter(SQLCmd, "UPDATE");
      } else if (SQLCmd.startsWith("DELETE")) {
         tableName = wordAfter(SQLCmd, "FROM");
      } else if (SQLCmd.startsWith("CREATE")) {
         tableName = wordAfter(SQLCmd, "TABLE");
      }
      return tableName;
   }

   protected Vector splitFields(String fields) {
      Vector fieldVector = new Vector();
      fields = fields.trim();
      if (fields.startsWith("(")) fields = fields.substring(1);
```

```
    if (fields.endsWith(")"))
      fields = fields.substring(0, fields.length() - 1);

    int comma = fields.indexOf(",");
    while (comma >= 0) {
      String field = fields.substring(0, comma).trim();
      fieldVector.addElement(field);
      fields = fields.substring(comma + 1).trim();
      comma = fields.indexOf(",");
    }
    fieldVector.addElement(fields.trim());
    return fieldVector;
  }

  protected Vector splitColumns(String columns) {
    return splitFields(columns);
  }

  protected Vector splitValues(String values) {
    return splitFields(values);
  }

  protected String wordAfter(String SQLCmd, String after) {
    String word = SQLCmd.substring(SQLCmd.indexOf(after) +
                        after.length()).trim();
    if (word.indexOf(" ")>-1) word=word.substring(0, word.indexOf(" "));
    return word.trim();
  }
}
```

② XMLQuery클라스

XMLQuery클라스는 기초클라스인 XMLCommand클라스를 확장한것이다. XMLQuery객체는 Statement.executeQuery()메쏘드가 호출될 때 XMLStatement에 의하여 창조된다. XMLQuery는 자기의 구축자에서 기초클라스의 parseSQLCmd()메쏘드를 호출한다. 목록 6-34는 XMLQuery클라스를 보여준다.

목록 6-34. XMLQuery클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;
```

```
import java.sql.SQLException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.xml.sax.InputSource;
import org.apache.xerces.parsers.DOMParser;
import java.sql.ResultSet;

public class XMLQuery extends XMLCommand {
  private Document xmlDoc;

  public XMLQuery(String SQLString) {
    this.SQLString = SQLString.toUpperCase().trim();
    parseSQLCmd(SQLString);
  }

  // XML문서를 처리하고 결과문서를 구축
  public XMLResultSet processDoc(Document xmlDoc) throws SQLException {
    this.xmlDoc = xmlDoc;
    XMLResultSet resultSet = new XMLResultSet();
    NodeList records = xmlDoc.getElementsByTagName(this.tableName);

    if (where == null) {
      for (int i = 0; i < records.getLength(); i++) {
        Element record = (Element) records.item(i);
        Node importedNode = resultSet.xmlDoc.importNode(record, true);
        if (!fields.equals("*")) pruneFields(importedNode);

resultSet.xmlDoc.getDocumentElement().appendChild(importedNode);
      }
    } else {
      Vector whereClauses = splitWhereClause(where);
      XMLWhereEvaluator evaluator = new XMLWhereEvaluator(whereClauses);

      for (int i = 0; i < records.getLength(); i++) {
        Element record = (Element) records.item(i);
        if (evaluator.testRecord(record)) {
          Node importedNode = resultSet.xmlDoc.importNode(record, true);
          if (!fields.equals("*")) pruneFields(importedNode);

resultSet.xmlDoc.getDocumentElement().appendChild(importedNode);
        }
```

```
      }
    }
    return resultSet;
  }

  protected Vector splitWhereClause(String whereClause) {
    Vector where = new Vector();

    String subTest = "";
    String token = "";
    StringTokenizer st = new StringTokenizer(whereClause, " ()", true);

    while (st.hasMoreTokens()) {
      token = st.nextToken();
      if (token.equals("AND") || token.equals("OR") ||
        token.equals("(") || token.equals(")")) {
        subTest = subTest.trim();
        if (subTest.length() > 0) where.addElement(subTest);
        where.addElement(token);
        subTest = "";
      } else {
        subTest += token;
      }
    }

    if (subTest.trim().length() > 0) {
      where.addElement(subTest.trim());
    }
    return where;
  }

  private void pruneFields(Node record) {
    Vector fieldClauses = splitFields(fields);
    NodeList nodes = record.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++) {
      Node n = nodes.item(i);
      if (n.getNodeType() == Node.ELEMENT_NODE) {
        String tagName = ((Element) n).getTagName();
        if (!fieldClauses.contains(tagName)) record.removeChild(n);
      }
    }
  }
}
```

XMLQuery객체를 만든 다음 XMLStatement는 질문이 진행되는 문서에 대한 참조를 넘겨주면서 XMLQuery클라스의 processDocument()메쏘드를 호출한다.

XMLQuery.processDocument()메쏘드는 질문의 실제적인 처리를 담당한다. 먼저 XMLResultSet를 만든 다음 표에 대응하는 XML요소들을 검색하고 그것들을 WHERE문절과 대조하여 평가한다.

자료기지가 XML문서에 포함되여있으므로 XMLResultSet도 역시 XML문서로 돌려진다. WHERE문절과 부합되는 XML요소들은 새로 만들어 문서에 들어가며 SQL질문의 렬목록에 항목별로 들어있지 않는 요소마디들은 잘라버린다. 한편 이 실현에서 속성마디들은 그대로 돌려진다. 물론 필요하다면 이것을 쉽게 변화시킬수 있다.

마지막 단계는 선택되여 정리된 마디를 XMLResultSet의 뿌리요소에 추가하는것이다. 일단 전체 XMLResultSet가 만들어지기만 하면 보통의 방법으로 귀환된다.

③ XMLWhereEvalutor클라스

SQL질문엔진자체는 뒤에서 보게 되는 XMLWhereEvalutor클라스에 실현되여있다. 보호된 요소레코드는 현재 검사되고있는 레코드를 포함하며 벡토르 testVector는 개별적인 부분검사조건들을 표시하는 문자렬들을 포함하고있다. 실례로 SQL질문이 다음과 같다고 하자.

```
"SELECT * FROM CUSTOMER
   WHERE (PERSONAL_NAME LIKE '성%' OR CUSTOMER_ID = '102')"
```

이때 WHERE문절은 아래에서와 같은 부분검사조건들로 분리된다.

```
(
PERSONAL_NAME LIKE '성%'
OR
CUSTOMER_ID = '102'
)
```

XMLWhereEvaluator는 결과문자렬을 만들기 위하여 XML문서의 매 행요소와 대조하여 검사벡토르를 평가한다. 결과문자렬은 검사문자렬에 괄호와 AND나 OR와 같은 론리연산자들을 직접 추가하고 연산자들을 포함하고있는 부분검사조건들을 평가하여 만든다. 부분검사조건들에 대한 평가는 그 부분검사조건의 연산자에 해당한 검사메쏘드를 호출하여 진행한다. 추가적인 검사를 진행하는것이 아주 간단하지만 여기서는 두가지 검사메쏘드들만 실현한다.

- isLike(): 통용문자 "%"를 분석하고 적당한 문자렬비교를 진행한다.
- isEqual(): 단순히 문자렬들을 비교한다.

이 검사메쏘드들은 결과를 론리값으로 돌려준다. 실례로 WHERE문절이 다음의 요소를 포함하고있는 어떤 행에 대하여 평가한다고 하자.

```
<PERSONAL_NAME>성철</PERSONAL_NAME>
```

그러면 귀환되는 결과문자렬은 다음과 같이 될것이다.

```
( true 혹은 false )
```

괄호안에 있는 결과문자렬은 evaluate(String infix)메쏘드에 넘겨진다. 이 메쏘드는 목록 6-35에서 보여준것처럼 결과문자렬을 평가하고 종합적인 검사에 대한 론리결과를 돌려주기 위하여 간단한 두개의 탄창메쏘드를 리용한다.

목록 6-35. XMLWhereEvalutor클라스

```
package JavaDB_Bible.ch06.sec03.JDBCforXML;

import java.io.*;
import java.util.*;
import java.sql.SQLException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.xml.sax.InputSource;
import org.apache.xerces.parsers.DOMParser;

public class XMLWhereEvaluator {
  Element record = null;
  Vector testVector = null;

  public XMLWhereEvaluator(Vector testVector) {
    this.testVector = testVector;
  }

  public boolean testRecord(Element record) {
    String test;
    String results = "";
    for (int i = 0; i < testVector.size(); i++) {
      test = (String) testVector.elementAt(i);
      if (test.equals("OR") || test.equals("AND") ||
          test.equals("(") || test.equals(")")) {
          results += " " + test;
      } else {
```

```
        if (testWhereClause(record, test)) results += " true";
        else results += " false";
      }
    }
    return evaluate(results.trim());
  }

  // 개별적인 where문절들을 검사
  private boolean testWhereClause(Element record,
                  String whereClause) {
    boolean not = false;
    boolean retval = false;
    String fieldName =
        whereClause.substring(0, whereClause.indexOf(" ")).trim();
    whereClause = whereClause.substring(fieldName.length()).trim();
    String test = whereClause.substring(0, whereClause.indexOf(" ")).trim();
    if (test.equals("NOT")) {
      not = true;
      whereClause = whereClause.substring(test.length()).trim();
      test = whereClause.substring(0, whereClause.indexOf(" ")).trim();
    }
    String operand = whereClause.substring(test.length()).trim();
    operand = operand.replace('\'', ' ').trim();
    String nodeValue = record.getAttribute(fieldName);
    if (nodeValue.length() == 0) {
      NodeList fields = record.getElementsByTagName(fieldName);
      Element field = (Element) fields.item(0);
      nodeValue = field.getFirstChild().getNodeValue();
    }

    if (test.equals("LIKE")) {
      retval = isLike(operand, nodeValue);
    }
    if (test.equals("=")) {
      retval = isEqual(operand, nodeValue);
    }
    if (test.equals("<>")) {
      not = true;
      retval = isEqual(operand, nodeValue);
    }
    if (not) retval = !retval;
    return retval;
  }

  private boolean isEqual(String operand, String nodeValue) {
    boolean retval = false;
```

```
    operand = operand.trim();
    if (nodeValue.equals(operand)) retval = true;
    return retval;
  }

  private boolean isLike(String operand, String nodeValue) {
    boolean retval = false;
    if (operand.startsWith("%")) {
      if (operand.endsWith("%")) {
        operand = operand.replace('%', ' ').trim();
        if (nodeValue.indexOf(operand) > -1) retval = true;
      } else {
        operand = operand.replace('%', ' ').trim();
        if (nodeValue.endsWith(operand)) retval = true;
      }
    } else if (operand.endsWith("%")) {
      operand = operand.replace('%', ' ').trim();
      if (nodeValue.startsWith(operand)) retval = true;
    } else {
      operand = operand.trim();
      if (nodeValue.equals(operand)) retval = true;
    }
    return retval;
  }

  protected boolean evaluate(String infix) {
    int parens = 0;
    Stack ops = new Stack();
    Stack args = new Stack();
    infix = infix.trim();

    StringTokenizer st = new StringTokenizer(infix, " ()", true);
    while (st.hasMoreTokens()) {
      String token = st.nextToken();
      if (!token.equals(" ")) {
        if (token.equals("AND") || token.equals("OR")) {
          if (ops.size() > parens) evaluate(ops, args);
          ops.push(token);
        } else if (token.equals("(")) {
          if (args.size() > 0)++parens;
        } else if (token.equals(")")) {
          --parens;
        } else {
          args.push(token);
        }
      }
```

```
    } while (!ops.empty()) {
      evaluate(ops, args);
    }

    String result = (String) args.pop();
    return (result.equals("true")) ? true : false;
  }

  private void evaluate(Stack ops, Stack args) {
    boolean a = (((String) args.pop()).equals("true")) ? true : false;
    boolean b = (((String) args.pop()).equals("true")) ? true : false;
    boolean c = false;

    String o = (String) ops.pop();
    if (o.equals("AND")) c = a & b;
    if (o.equals("OR")) c = a | b;
    args.push(c ? "true" : "false");
  }
}
```

6.3.3. JDBC/XML자료기지에 대한 검사

앞에서 본 임의의 DriverManager에 기초한 실례들과 류사한 코드를 써서 JDBC/XML자료기지를 검사할수 있다. 목록 6-36에서는 흘리기가능한 ResultSet의 모든 기능들을 리용하는 전형적인 실례를 보여준다. 이 실례는 목록 6-31의 흘리기가능한 ResultSet의 메쏘드들을 목록 6-30의 XMLResultSet에 추가하여 실현된다.

목록 6-36. JDBC/XML자료기지검사코드

```
package JavaDB_Bible.ch06.sec03;

import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
import org.w3c.dom.Document;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import JavaDB_Bible.ch06.sec03.JDBCforXML.XMLResultSet;
import JavaDB_Bible.ch06.sec03.JDBCforXML.XMLStatement;

public class XMLDBTest {
   static String UrlString = "file:///d:/sample/CustomerDB.xml";

   static String SQLQuery = "SELECT * FROM CUSTOMER WHERE " +
```

```
                    "(PERSONAL_NAME LIKE '성%' OR CUSTOMER_ID = '102') " +
                    "AND FAMILY_NAME = '김'";*/

    static String[] SQLCmd = {
                       "INSERT INTO CUSTOMER VALUES('101',김','은희','녀'," +
"'평양시','평양','보통강구역','대보동','17','453-5521')",
                       "INSERT INTO CUSTOMER VALUES('102','김','철','남'," +
                       "'함경남도','함흥',NULL,'사포동','32',NULL)"};

    public String cID = null;
    public String fName = null;
    public String pName = null;
    public String sex = null;
    public String state = null;
    public String city = null;
    public String district = null;
    public String dong = null;
    public String neighbor = null;
    public String phone = null;

    Document xmlDoc = null;

    public XMLDBTest() {
        try {
Class.forName("JavaDB_Bible.ch06.sec03.JDBCforXML.XMLDriver");
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public static void main(String args[]) {
        XMLDBTest test = new XMLDBTest();
        serializeDocumentAsFile(test.createTable(), UrlString);
        serializeDocumentAsFile(test.updateTable(SQLCmd), UrlString);
        serializeDocumentAsFile(test.queryTable(SQLQuery),
                              "file:///d:/sample/ResultSet.xml");
    }

    public Document createTable() {
        try {
            Connection con = DriverManager.getConnection(UrlString);
            Statement stmt = con.createStatement();

            stmt.executeUpdate("CREATE TABLE CUSTOMER " +
```

```
                        "(CUSTOMER_ID ATTRIBUTE, " +
                        "FAMILY_NAME VARCHAR(30), " +
                        "PERSONAL_NAME VARCHAR(30), " +
                        "SEX VARCHAR(4), " +
                        "STATE VARCHAR(30), " +
                        "CITY VARCHAR(30), " +
                        "DISTRICT VARCHAR(30), " +
                        "DONG VARCHAR(30), " +
                        "NEIGHBOR VARCHAR(30), " +
                        "PHONE VARCHAR(30))");

        stmt.executeUpdate("INSERT INTO CUSTOMER VALUES(" +
                        "'100','김','성철','남','평양시','평양'," +
                        "'중구역','교구동','32','326-6532')");

        xmlDoc = ((JavaDB_Bible.ch06.sec03.JDBCforXML.XMLStatement)stmt).
                getXmlDocument();
    } catch (Exception e) {
        System.out.println(e);
    }
    return xmlDoc;
}

public Document updateTable(String[] SQLCmd) {
    try {
        Connection con = DriverManager.getConnection(UrlString);
        Statement stmt = con.createStatement();
        for (int i = 0; i < SQLCmd.length; i++) {
            stmt.executeUpdate(SQLCmd[i]);
        }
        xmlDoc = ((JavaDB_Bible.ch06.sec03.JDBCforXML.XMLStatement)stmt).
                getXmlDocument();
    } catch (Exception e) {
        System.out.println(e);
    }
    return xmlDoc;
}

public Document queryTable(String SQLQuery) {
    ResultSet rs = null;
    try {
        Connection con = DriverManager.getConnection(UrlString);
        Statement stmt = con.createStatement();

        rs = stmt.executeQuery(SQLQuery);
        while (rs.next()) {
```

```
                getRowData(rs);
            }
            while (rs.previous()) {
                getRowData(rs);
            }

            rs.first();
            getRowData(rs);

            rs.last();
            getRowData(rs);

            rs.absolute(2);
            getRowData(rs);

            rs.relative( -1);
            getRowData(rs);
        } catch (Exception e) {
            System.out.println(e);
        }
        return ((JavaDB_Bible.ch06.sec03.JDBCforXML.XMLResultSet) rs).xmlDoc;
    }

    private void getRowData(ResultSet rs) {
        try {
            cID = rs.getString("CUSTOMER_ID");
            fName = rs.getString("FAMILY_NAME");
            pName = rs.getString("PERSONAL_NAME");
            sex = rs.getString("SEX");
            state = rs.getString("STATE");
            city = rs.getString("CITY");
            district = rs.getString("DISTRICT");
            neighbor = rs.getString("NEIGHBOR");
            dong = rs.getString("DONG");
            phone = rs.getString("PHONE");
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public static void serializeDocumentAsFile(Document xmlDoc,
                                                String UrlString) {
        String fileName = "XMLOut.xml";
        try {
            URL url = new URL(UrlString);
            if (url.getProtocol().equals("file")) {
```

```
                fileName = url.getFile().substring(1);
            }

            OutputFormat fmt = new OutputFormat("xml", null, true);
            XMLSerializer serializer =
                    new XMLSerializer(new FileWriter(fileName), fmt);
            serializer.asDOMSerializer().serialize(xmlDoc);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

이 검사코드는 목록 6-37에 보여준 XML자료기지를 창조한다. createTable()메쏘드는 XML문서를 창조하고 첫 레코드를 삽입한다. updateTable()메쏘드의 호출은 다른 레코드들을 삽입한다.

목록 6-37. XMLDB검사클라스를 리용하여 창조한 XML자료기지

```
<?xml version="1.0"?>
<CustomerDB>
   <CUSTOMER CUSTOMER_ID="100">
      <FAMILY_NAME>김</FAMILY_NAME>
      <PERSONAL_NAME>성철</PERSONAL_NAME>
      <SEX>남</SEX>
      <STATE>평양시</STATE>
      <CITY>평양</CITY>
      <DISTRICT>중구역</DISTRICT>
      <NEIGHBOR>교구동</NEIGHBOR>
      <DONG>32</DONG>
      <PHONE>326-6532</PHONE>
   </CUSTOMER>
   <CUSTOMER CUSTOMER_ID="101">
      <FAMILY_NAME>김</FAMILY_NAME>
      <PERSONAL_NAME>은희</PERSONAL_NAME>
      <SEX>녀</SEX>
      <STATE>평양시</STATE>
      <CITY>평양</CITY>
      <DISTRICT>보통강구역</DISTRICT>
      <NEIGHBOR>대보동</NEIGHBOR>
      <DONG>17</DONG>
```

```
        <PHONE>453-5521</PHONE>
    </CUSTOMER>
    <CUSTOMER CUSTOMER_ID="102">
        <FAMILY_NAME>김</FAMILY_NAME>
        <PERSONAL_NAME>철</PERSONAL_NAME>
        <SEX>남</SEX>
        <STATE>함경남도</STATE>
        <CITY>함흥</CITY>
        <DISTRICT></DISTRICT>
        <NEIGHBOR>사포동</NEIGHBOR>
        <DONG>23</DONG>
        <PHONE></PHONE>
    </CUSTOMER>
</CustomerDB>
```

SQL의 CREATE지령에서는 대부분의 마당들에 자료형을 VARCHAR(30)로 지정하지만 여기서는 기정으로 String을 주고있다. 그 리유는 모든 자료가 문자렬로 저장되며 자료형에 붙인 의미는 오직 전용자료형 ATTRIBUTE에 대한 검사뿐이기때문이다. ATTRIBUTE 자료형은 그 마당이 행요소에 속성으로 추가되여야 한다는것을 지시하는데 리용된다.

또한 XML문서는 매번 갱신된 다음에는 보관되여야 한다. 실제상 XML자료기지는 기억기상에 DOM문서로 존재하며 따라서 변경된 다음에는 직렬화되여야 한다.

검사는 각이한 질문들을 리용하여 수행된다. 이 질문들에는 다음과 같은것들이 포함된다.

```
SELECT * FROM CUSTOMER
SELECT * FROM CUSTOMER
SELECT * FROM CUSTOMER WHERE PNAME LIKE '은%'
SELECT * FROM CUSTOMER WHERE PNAME NOT LIKE '은%'
SELECT * PROM CUSTOMER WHERE FNAME NOT = '김'
SELECT * FROM CUSTOMER WHERE FNAME <> '김'
SELECT * FROM CUSTOMER WHERE PNAME LIKE '은%' OR PNAME LIKE '성%'
SELECT * FROM CUSTOMER WHERE (PNAME LIKE '은%' OR PNAME LIKE '철%')
SELECT * FROM CUSTOMER WHERE (PNAME LIKE '은%' OR CUSTOMER_ID = '102')
```

XMLResultSet는 목록 6-36에서 String변수들을 설정하는데 리용되는 ResultSet.getString()메소드를 지원하는외에 XML문서로 검색될수도 있다. 목록 6-38은 다음의 질문을 실행하여 만든 XMLResultSet를 보여준다.

```
"SELECT * FROM CUSTOMER
WHERE (PERSONAL_NAME LIKE '성%' OR CUSTOMER_ID = '102') AND
      FAMILY_NAME = '김'
```

목록 6-38. XMLResultSet

```
<?xml version="1.0"?>
<RESULTSET>
  <CUSTOMER CUSTOMER_ID="100">
    <FAMILY_NAME>김</FAMILY_NAME>
    <PERSONAL_NAME>성철</PERSONAL_NAME>
    <SEX>남</SEX>
    <STATE>평양시</STATE>
    <CITY>평양</CITY>
    <DISTRICT>중구역</DISTRICT>
    <NEIGHBOR>교구동</NEIGHBOR>
    <DONG>32</DONG>
    <PHONE>326-6532</PHONE>
  </CUSTOMER>
  <CUSTOMER CUSTOMER_ID="102">
    <FAMILY_NAME>김</FAMILY_NAME>
    <PERSONAL_NAME>철</PERSONAL_NAME>
    <SEX>남</SEX>
    <STATE>함경남도</STATE>
    <CITY>함흥</CITY>
    <DISTRICT></DISTRICT>
    <NEIGHBOR>사포동</NEIGHBOR>
    <DONG>23</DONG>
    <PHONE></PHONE>
  </CUSTOMER>
</RESULTSET>
```

전체 XMLResultSet를 XML문서로 돌려주는 메쏘드는 아주 쓸모가 있다. 그것은 대부분의 프로그람들이 XML과 작업하도록 설계되여있기때문이다. XMLResultSet는 XML형식으로 프로그람들사이에 전송되거나 XSL변환을 리용하여 처리될수 있다.

실례에서 리용한 자료기지가 URL에 의하여 정의되였기때문에 국부XML파일들은 물론 원격콤퓨터에 있는 파일들에 대해서도 리용할수 있다.

6장에 대한 요약

이 장에서는 자료기지와 XML을 함께 리용하는 방법을 고찰하였다. 표로부터 자료를 얻고 결과를 XML로 형식화하였으며 인터네트에서 XML자료를 얻어 자료기지표에 보관하였다. 계속하여 JDBC `RowSet`와 SQL질문을 리용하여 XML문서에 접근하는 간단한 JDBC구동프로그람을 만들었다. 론의한 기본문제들은 다음과 같다.

- XML문서들을 분석하기 위한 DOM분석기의 리용
- Xbean을 리용한 XML문서의 작성과 처리
- 자료기지 질문에 의한 XML문서작성
- XML자료원천으로부터 표에 자료옮기기
- 접속형 `RowSet`와 비접속형 `RowSet`
- `CashedRowSet`와 `WebRowSet`의 리용
- JDBC구동프로그람의 세부적인 동작
- Stirng지향 SQL질문엔진
- XML문서들과의 작업실례

찾아보기

Java자료기지프로그람작성법

집　필　리률남, 한은정, 로순영　　심　사　김정훈
편　집　김철우　　교　정　서금석
장　정　서경애　　콤퓨터편성　여은정

낸　곳　교육성 교육정보쎈터　　인쇄소　교육성 교육정보쎈터
인　쇄　주체97(2008)년 8월 10일　　발　행　주체97(2008)년 8월 20일

교-07-1307